

NO-A191 357

A SENSOR WITH BIOLOGICAL PREPROCESSING FEATURES(U)  
VIRGINIA UNIV CHARLOTTESVILLE DEPT OF ELECTRICAL  
ENGINEERING R M INIGO ET AL 18 DEC 87

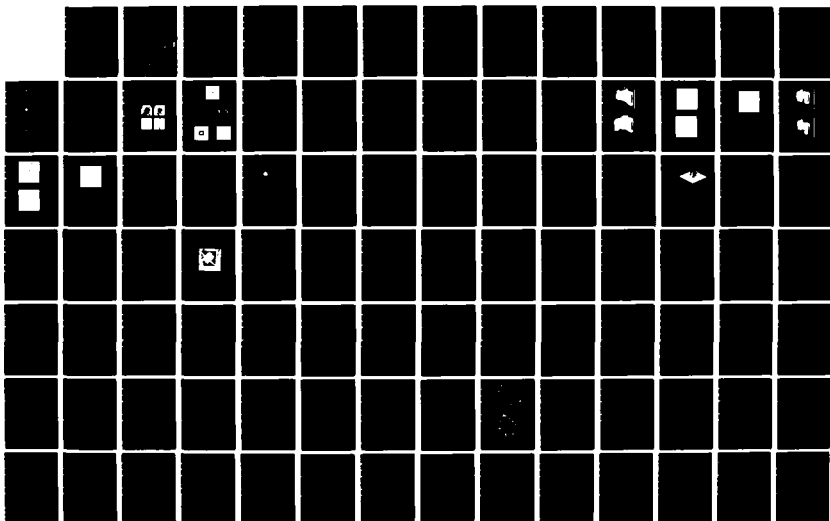
172

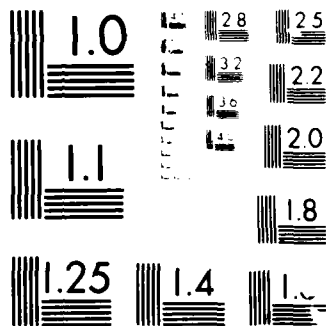
UNCLASSIFIED

UVA/525679/EE88/101 AFOSR-TR-88-0048

F/G 12/9

NL





MICROCOPY RESOLUTION TEST CHART  
 NATIONAL BUREAU OF STANDARDS-1963-A

AD-A191 357

2

DTIC FILE COPY

AFOSR-TR. 88-0048

A Final Report  
Grant No. AFOSR-85-07-0363-B  
September 1, 1985 - October 31, 1987

**A SENSOR WITH BIOLOGICAL PREPROCESSING FEATURES**

Submitted to:

Air Force Office of Scientific Research  
Building 412  
Bolling Air Force Base  
Washington, D.C. 20332-6448

Submitted by:

Rafael M. Inigo  
Professor

Eugene S. McVey  
Professor

Jonathan F. Doner  
Research Assistant Professor  
Biomedical Engineering

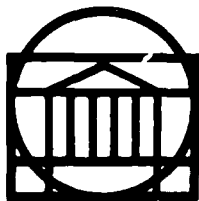
Chiewcharn Narathong  
Graduate Student

Jay I. Minnix  
Graduate Student

John Sigda  
Graduate Student

Report No. UVA/525679/EE88/101  
December 1987

DTIC  
ELECTE  
FEB 24 1988  
S D



**SCHOOL OF ENGINEERING AND  
APPLIED SCIENCE**

This document has been approved  
for public release and unless its  
distribution is restricted

88 2 24 120

**UNIVERSITY OF VIRGINIA  
CHARLOTTESVILLE, VIRGINIA 22901**

ADA191357

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1d. RESTRICTIVE MARKINGS None		
2a. SECURITY CLASSIFICATION AUTHORITY			2. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UVA/525679/EE87			5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR. 88-0048		
6a. NAME OF PERFORMING ORGANIZATION University of Virginia Dept. of Electrical Engineering		6b. OFFICE SYMBOL (If applicable) A		7a. NAME OF MONITORING ORGANIZATION AFOSR	
6c. ADDRESS (City, State, and ZIP Code) Thornton Hall Charlottesville, VA 22901			7b. ADDRESS (City, State, and ZIP Code) Building 410, Bolling Air Force Base Washington, DC 20332-6448		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Office of Scientific Research/NE		8b. OFFICE SYMBOL (If applicable) NE		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-85-06-0363-B	
8c. ADDRESS (City, State, and ZIP Code) Building 410 Bolling Air Force Base Washington, DC 20332-6448		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO. U1102F		PROJECT NO. 2305	TASK NO. B4
				WORK UNIT ACCESSION NO.	
11. TITLE (Include Security Classification) A Sensor with Biological Preprocessing Features					
12. PERSONAL AUTHOR(S) R. M. Inigo, E. S. McVey, J. F. Doner					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM 9/1/87 TO 10/31/87		14. DATE OF REPORT (Year, Month, Day) 18 December 1987	
15. PAGE COUNT 103					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This report discusses research on "A Sensor with Biological Preprocessing Features" during the period February 1, 1986 to October 31, 1987. The first part of the report reviews the basic geometric sensor configuration and its mapping to computation plane. Three basic configurations are discussed and their application to edge and short range motion detection follows. The next section covers a qualitative long range motion detection algorithm. The algorithm was tested first with synthetic images and next with real images, and in both cases the results were very satisfactory. The algorithm is now being tested with multiple images. A one line correlation tracker is discussed in the next section. The correlation tracker can be implemented with parallel architecture and is, thus, very fast. Tests have been performed with satisfactory results using real images. Pattern recognition using the BVS together with a class of transforms which are invariant to translation is discussed. It should be possible					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. C. Lee Giles			22b. TELEPHONE (Include Area Code) (202) 767-4931		22c. OFFICE SYMBOL NE

DD FORM 1473, 84 MAR

33 APR edition may be used until exhausted.  
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

A

to implement this pattern recognition system using neural networks. A preliminary neural network implementation is given. The use of learning neural nets will allow recognition of similar but slightly distorted objects. A discussion on the implementation of the dual sensor follows. A hardware configuration is proposed to map a rectangular array into a log-spiral array; and an error analysis is performed.

Finally, depth computation from optical flow using the new sensor is discussed.

Keywords: cybernetics; image processing; spatial distribution;  
two dimensional; three dimensional; computer simulation

A Final Report  
Grant No. AFOSR-85-03-0363-B  
September 1, 1985 - October 31, 1987

A SENSOR WITH BIOLOGICAL PREPROCESSING FEATURES

Submitted to:  
Air Force Office of Scientific Research  
Building 412  
Bolling Air Force Base  
Washington, D.C. 20332-6448  
Attention: Dr. C. Lee Giles

Submitted by:  
Rafael M. Inigo  
Professor  
Eugene S. McVey  
Professor

Jonathon F. Doner  
Research Assistant Professor  
Biomedical Engineering

Chiewcharn Narathong  
Graduate Student

Jay Minnix  
Graduate Student

Zia-ur Rahman  
Graduate Student

William I. Clement  
Graduate Student

John Sigda  
Graduate Student

Department of Electrical Engineering  
SCHOOL OF ENGINEERING AND APPLIED SCIENCE  
UNIVERSITY OF VIRGINIA  
CHARLOTTESVILLE, VIRGINIA

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input checked="checked" type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Report No. UVA/525679/EE88/101

Copy No. \_\_\_\_\_

## TABLE OF CONTENTS

<b>1. INTRODUCTION .....</b>	<b>1</b>
<b>2. REVIEW OF PREVIOUS WORK .....</b>	<b>3</b>
2.1. Biological Visual Sensor Configuration .....	3
2.2. Edge and Motion Detection With the New Sensor .....	3
<b>3. LONG RANGE QUALITATIVE MOTION DETECTION ALGO- RITHM .....</b>	<b>9</b>
3.1. Algorithm Testing .....	10
3.2. Problems with Real Images .....	11
3.3. Schemes for Preprocessing Images .....	11
3.4. Examples and Further Discussion .....	12
<b>4. A FAST ALGORITHM FOR MOTION PREDICTION .....</b>	<b>21</b>
4.1. Introduction .....	21
4.2. The Tracking Algorithm .....	21
4.2.1. 2D Motion (a general case) .....	27
4.2.2. Simulation with a Gaussian Grey Level Image .....	27
4.3. Translation, Rotation, and Scaling .....	31
4.3.1. Simulations with a Real Image .....	34
4.4. Speed Comparison .....	36
4.4.1. Conventional Correlation .....	36
4.4.2. One-Dimensional Correlation .....	42
4.5. Conclusion .....	43

<b>5. PATTERN RECOGNITION USING THE BVS, THE C-TRANSFORM AND A NEURAL NETWORK CLASSIFIER .....</b>	<b>46</b>
5.1. Introduction .....	46
5.2. Background .....	46
5.3. R and M Transform Advantages and Disadvantages .....	49
5.4. Computer Simulation Results .....	50
5.5. Simulation of Two-Dimensional Case .....	50
5.6. Use of Transforms With BVS Sensor .....	51
5.7. Network Structure and Weight Matrix .....	52
5.8. Reduced Transform Representation .....	54
5.9. Extension into Two Dimensions .....	55
5.10. <i>Implementation of Learning in an ANS</i> .....	56
<b>6. DUAL SENSOR IMPLEMENTATION AND ANALYSIS .....</b>	<b>64</b>
6.1. Introduction .....	64
6.2. Sensor Technology .....	64
6.3. Method for Obtaining Dual Representations of Images .....	66
6.4. A Method to Determine which Solution is Best .....	67
6.5. Analysis of Error for Possible Solutions .....	69
6.6 A Possible Hardware Implementation of Single Camera/Dual Sensor .....	75
6.6.1 Operation of the Analog Summing Arrangement .....	75
6.6.2 Operation of the Digital Summing Arrangement .....	75
6.7 Error Analysis .....	79

<b>7. DEPTH COMPUTATION FROM OPTIC FLOW .....</b>	<b>86</b>
7.1. Introduction .....	86
7.2. Overview of Current Research .....	86
7.3. Computation of Optic Flow .....	87
7.4. Physiological Correlates of Optic Flow Computation .....	88
7.5. An Approach to 3D Computation from EM .....	88
<b>8. VESTIBULAR-OCULAR MOTION FOR TARGET TRACKING .....</b>	<b>91</b>

#### **APPENDIX 6.A : ERROR GRAPHS**

## LIST OF FIGURES

Fig. 2.1(a) Image and Computation Planes. Arc-of-ring. ....	4
Fig. 2.1(b) Image and Computation Planes. Circular elements ....	4
Fig. 2.1(c) Image and Computation Planes. Hexagonal elements ....	4
Fig. 2.2 Edge Detection Using BVS .....	6
Fig. 2.3(a) Original image: a noisy image of a lady's face .....	6
Fig. 2.3(b) Simulated circular-element sensor image, image plane .....	6
Fig. 2.3(c) Edge detection by Laplacian-Gaussian method, with thresholding, comp. plane .....	6
Fig. 2.3(d) Edge detection by Laplacian method, with thresholding, comp. plane .....	6
Fig. 2.4 Motion Detection by the BVS .....	7
Fig. 2.5 Short-range Motion Detection .....	7
Fig. 2.6 Motion Detection by the New Sensor	
(a) A circle inside a square .....	7
(b) The detection motion of a circle and a square .....	7
Fig. 3.1 Car in initial and moved position .....	14
Fig. 3.2 Car in initial and moved position after preprocessing .....	15
Fig. 3.3 Difference Image .....	16
Fig. 3.4 Car in initial and moved positions .....	17
Fig. 3.5 Image after preprocessing .....	18
Fig. 3.6 Difference image .....	19
Fig. 4.1 The nonuniform logarithmic spiral sensor, circular elements .....	22
Fig. 4.2 Row Search Mechanism .....	28
Fig. 4.3 Mixed density Gaussian grey level image .....	29
Fig. 4.4 Histogram for correlation of rows $k$ and $k+i$ .....	29

## LIST OF TABLES

Table 3.1	Motion to the left and down .....	13
Table 3.2	Motion to the right and down .....	19
Table 4.I	Row correlation for search area, a Gaussian image .....	30
Table 4.II	A summary of horizontal and vertical perturbations for a Gaussian image .....	32
Table 4.III	More simulations for a Gaussian image .....	32
Table 4.IV	A real image with $\Delta x=5.0$ , $\Delta y=-4.0$ , $\Delta \theta=11.0^\circ$ , $\kappa=.84$ .....	37
Table 4.V(a)	First iteration $\Delta x=5.0$ , $\Delta y=-4.0$ , $\Delta \theta=11.0^\circ$ , $\kappa=.84$ .....	39
Table 4.V(b)	First iteration $\Delta x=5.0$ , $\Delta y=-4.0$ , $\Delta \theta=11.0^\circ$ , $\kappa=.84$ .....	39
Table 4.VI(a)	Second iteration $\Delta x=5.0$ , $\Delta y=-4.0$ , $\Delta \theta=11.0^\circ$ , $\kappa=.84$ .....	40
Table 4.VI(b)	Second iteration $\Delta x=5.0$ , $\Delta y=-4.0$ , $\Delta \theta=11.0^\circ$ , $\kappa=.84$ .....	40
Table 4.VII	A summary of estimates in the rectangular plane of a real image .....	41

## 1. INTRODUCTION

This final report covers research done on "A Sensor with Biological Preprocessing Features" under the sponsorship of the Air Force Office of Scientific Research, covering the period February 1, 1986 to September 30, 1987.

As a result of the effort, four students have obtained the Master of Science degree in Electrical Engineering and one student the Doctor of Philosophy degree in Electrical Engineering [1,2,3,4,5]. Eleven papers have been written, five of which have been published [6,7,8,9,10], one is an invited paper to be published shortly [11], one will be published shortly [12], and the rest are under the process of review [13,14,15,16].

Research results obtained through January 1986 were reported in the Final Report on Grant No. AFOSR-84-0349, "Biological Visual Systems Structures for Machine Vision Applied to Robotics", Report No. UVA/525647/EE86/101, February 1986 [41]. A brief summary complemented with results not reported in that report follows in Section 2. The main body of this final report contains more recent results. The report is organized in the order listed below. For clarity and convenience, references have been listed at the end of each section.

1. Introduction
2. Review of Previous Work
3. Long Range Qualitative Motion Detection Algorithm
4. A Fast Algorithm for Motion Prediction
5. Pattern Recognition Using the BVS, the C-transform and a Neural Network Classifier
6. Dual Sensor Implementation and Analysis
7. Depth Computation from Optical Flow
8. Vestibular-ocular Motion for Target Tracking

## REFERENCES

### A. Dissertations and Thesis

1. C. Narathong, "An Algorithm for Motion Prediction Using a Biological Visual Sensor"
2. C. Hsin, "Edge and Motion Detection Using Peripheral Human Visual Properties," Master of Science Thesis, University of Virginia, May 1986.
3. J. I. Minnix, "Structural Analysis and Design of a Biological Based Vision Sensor," M.S. Thesis, University of Virginia, August 1986.
4. Z. Rahman, "Motion Detection Using a Biological Type Vision Sensor," M.S. Thesis, University of Virginia, August 1986.
5. V. L. Davis, "Misrosaccadic Eye Movement and its Applications to Sensor Resolution Enhancement," M.S. Thesis, University of Virginia, May 1987.

## B. Papers

6. Minnix, J. I., Iñigo R. M., and McVey, E. S., "A New Sensor for Machine Vision," Proc. of Asilomar Conference, Monterey, CA, November 1985.
7. Iñigo, R. M. et al., "A Biological-structure Visual Sensor for Robotics Applications," Proc. 16th International Symp. on Industrial Robots, Brussels, Belgium, Sept. 1986.
8. Iñigo, R. M. et al., "A New Sensor for Machine Vision," Proc. International IFAC Conf. on Low Cost Sensors, Valencia, Spain, October 1986.
9. Narathong et al., "An Algorithm for Motion Prediction Using a Biological Vision Sensor," IEEE International Conf. on Robotics and Automat., Raleigh, NC, March 1987.
10. McVey, E. S. et al., "Fault Tolerant Array Sensor Design," IEEE Southeastcon '87, Tampa, FL, April 1987.
11. Narathong, C., Iñigo, R. M. et al., "A Target Tracking Algorithm Using A Sensor with Biological Vision Features," invited paper, submitted to International Journal of Machine Tools and Manufacture, for publication in the Special Supplement on Robotics and Artificial Intelligence, 1987.
12. McVey, E. S. and Iñigo, R. M., "Static Design and Constraints of Analog Neural Networks," Proc. Twenty-first Annual Asilomar Conf. on Signals, Systems, and Computers, Pacific Grove, CA, Nov. 2-4, 1987.
13. Iñigo, R. M., Narathong, C. et al., "A Fast Algorithm for Motion Prediction," submitted to IEEE PAMI for review, April, 1987.
14. Hsin, C. and Iñigo, R. M., "Edge and Motion Detection Using Properties of the Human Peripheral Visual System," submitted to CVGIP for review, April, 1987.
15. Clement, W. I. et al., "Synaptic Strengths For Neural Simulation of the TSP," submitted to IEEE Trans. on Circuits and Systems for review, April, 1987.
16. Iñigo, R. M. et al., "A Vision System With Biological Features for Robotics," submitted to 18th symposium on International Robots.

## 2. REVIEW OF PREVIOUS WORK

### 2.1. Biological Visual Sensor Configuration

Three basic configurations for the image plane were considered: arcs of ring (called "rectangular elements"), circular elements and hexagonal elements, see Fig. 2.1. Design rules relating the number of concentric rings to the number of elements per ring for the three configurations were developed. In addition, a program to simulate images that would be obtained with these structures, using real images obtained with a conventional 512 by 512 CCD camera, was written and used for all subsequent work. Based on the field of view covered by the sensor for a specific optic configuration, the number of pixels of the log-spiral sensor is orders of magnitude smaller than that of a conventional rectangular sensor, for equivalent pixel size in the fovea region. The basic properties of the BVS sensor configuration, such as invariance to scaling and rotation about the optical axis were verified using real images.

Additional work not included in the report has been done on the use of chain coding for the six neighborhood computation plane grid. Several binary images were chain-coded for comparison with eight neighborhood coding of conventional images. Due to the fact that the six neighborhood coding was applied in the computation plane, where images have a completely different shape than in the image plane, the chain-length was not as short as the reduction in number of pixel would seem to indicate, but it was significantly shorter than that of the conventional sensor images, nevertheless. In addition, the process is faster because only six directions are used. Once the images were chain coded, pattern recognition was performed on them using chain-coded template matching which is more convenient than other methods in this case. For example, a circle has only one chain direction and chain length, irrespective of size, when it is centered on the optical axis and the code does not change for other centered objects either, irrespective of size and rotation.

### 2.2. Edge and Motion Detection with the New Sensor

This part of the research dealt modelling properties of the human peripheral visual system (HPVS) for edge and motion detection and its application to the new sensor. A good summary of outstanding research results by other investigators in the area of modelling is given in the February 1986 report and in [1].

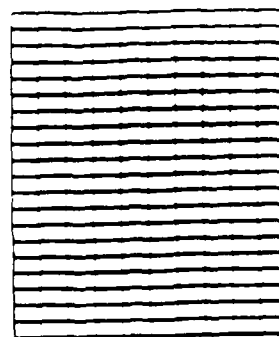
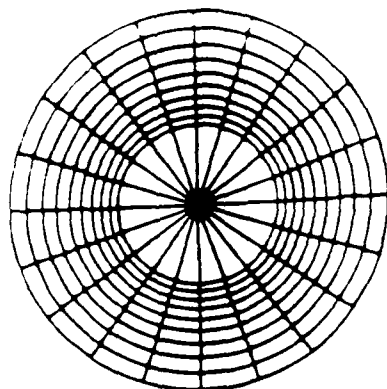
Edge detection for the arcs-of-ring sensor can be implemented in the computation-plane's rectangular grid using any of the standard methods such as local neighborhood operator, Laplacian-Gaussian or global methods related to signal analysis [2, 3]. For the circular and hexagonal sensors, the computation-plane array is staggered, forming a hexagonal array and the following two neighborhood operators can be used:

$$g_A = |a_1 - a_2| + |a_1 - a_3| + |a_1 - a_4| + |a_1 - a_5| + |a_1 - a_6| + |a_1 - a_7| \quad (2.1)$$

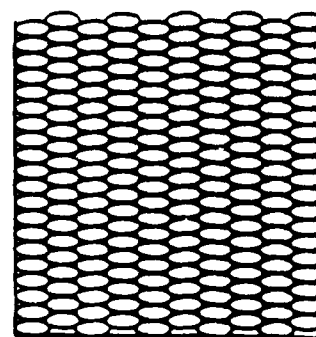
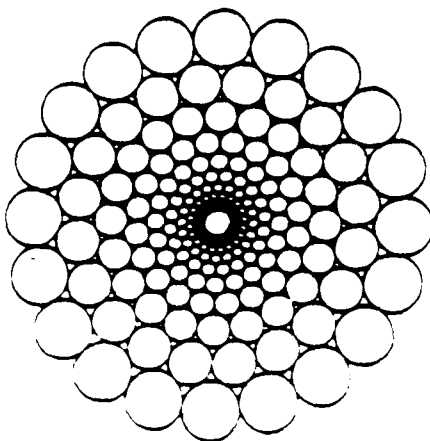
and

$$g_L = (a_1 - a_2) + (a_1 - a_3) + (a_1 - a_4) + (a_1 - a_5) + (a_1 - a_6) + (a_1 - a_7) \quad (2.2)$$

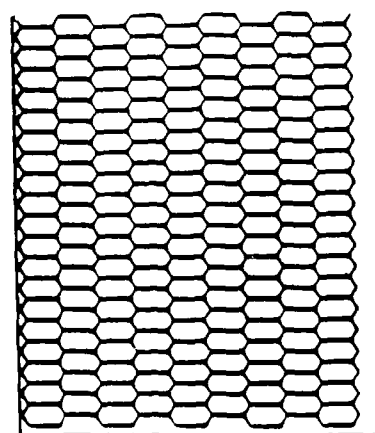
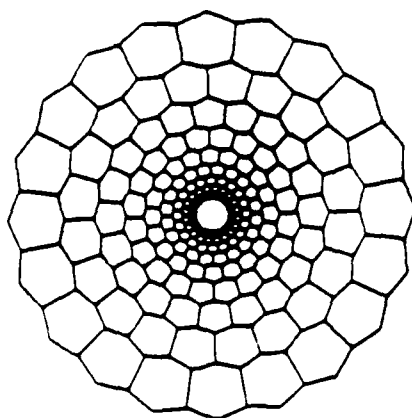
These are, respectively, the absolute value and the Laplacian methods. The absolute value method does not produce zero-crossings and is, consequently, less similar to



(a) Image and Computation Planes. Arc-of-ring.



(b) Image and Computation Planes. Circular elements.



(c) Image and Computation Planes. Hexagonal elements.

Fig. 2.1

HPVS models than methods such as the Laplacian-of-the-Gaussian. In the Laplacian method positive and negative gray level values around the regions of abrupt intensity change are obtained, and the zero-crossing points between these are chosen as the edges. In order to eliminate spurious edges in noisy images, an optional thresholding operator can be applied. The general method is shown in block diagram in Fig. 2.2. Simulations with synthetic and real images obtained with a 512 by 512 CCD camera and transformed to the new sensor equivalent image by the technique described in the report. These were performed using the Laplacian-Gaussian method and the two operators of formulas (2.1) and (2.2). For synthetic images, the results using (2.1) were practically identical to those of the Laplacian-Gaussian. For a real image, a noise photograph of a girl's face, Fig. 2.3(a), the results for (2.1) and the L-G are given in Figs. 2.3(b) and (c), respectively. Notice that the edges are very similar in both cases. These results correspond to a low resolution circular sensor of 36 rings, 75 elements per ring. More detailed edges are obtained with a higher resolution simulated sensor of 51 rings, 120 elements per ring.

For motion detection, the same approach was used as for edge detection. Biological motion detection was first reviewed [1] using the two process theory of HVS and then simulations were performed using the T and U channels [4] for short-range mode. The procedure for simulation was:

- a) Perform spatial convolution  $S(x, y, t) = \nabla^2 G * f(x, y, t)$
- b) Establish the orientation of zero-crossing contours
- c) Apply temporal derivation to  $S(x, y, t)$ :  $T(x, y, t) = \partial S(x, y, t) / \partial t$
- d) To detect motion at each zero-crossing, a positive value of T indicates motion towards the negative side of the crossing; a negative value of T indicates the opposite. The direction of motion is perpendicular to the edge.
- e) All the unit directional vectors in an edge region with a given orientation are combined into a single unit vector pointing in that direction. After this operation is performed, unit directional vectors in adjacent regions with different orientation are combined in the same way, proceeding along the edge, until the complete closed edge has been processed. This will give the direction of motion. The resolution is limited to  $45^\circ$  increment. In Fig. 2.4, a square is translated at an angle of  $45^\circ$ , as indicated by the arrow at the center of the square.

The simulation in short-range mode for the new detector is related to intensity based schemes in computer vision. The simulation procedure was as follows:

- a) Apply the six-neighborhood Laplacian operation to the input images.
- b) Establish the orientation of zero-crossings for the first frame and apply 7-element chain coding to the zero-crossing contour to segment the objects.
- c) Perform temporal derivation by subtraction,  $T(x, y, t) = S(x, y, t + \Delta t) - S(x, y, t)$ .
- d) With reference to Fig. 2.5, in which positively marked pixels correspond to high intensity at edge boundary and negatively marked ones to low intensity, when the edge is moving towards the negative side as shown in Figs. 2.5(a) and (d), all seven elements will become positive at the second frame. When the edge is moving towards the positive side, some elements may remain positive. The direction of local motion is perpendicular to edge orientation at zero-crossing and the displacement is limited to two pixels.

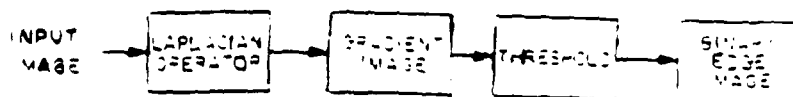
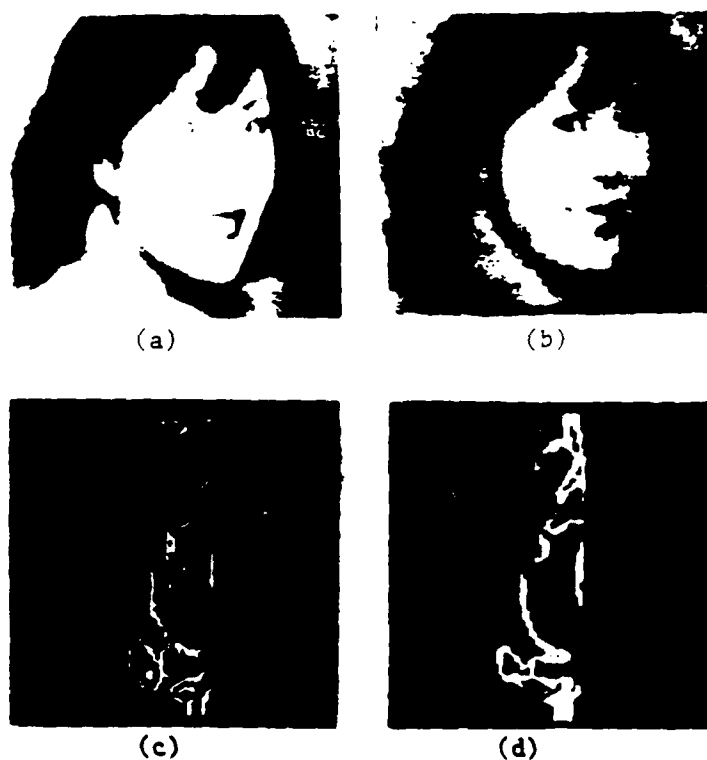


Fig. 2.2 Edge Detection Using BVS.



- (a) Original image: a noisy image of a lady's face.
- (b) Simulated circular-element sensor image, image plane.
- (c) Edge detection by Laplacian-Gaussian method, with thresholding, comp. plane.
- (d) Edge detection by Laplacian method, with thresholding, comp. plane.

Fig. 2.3

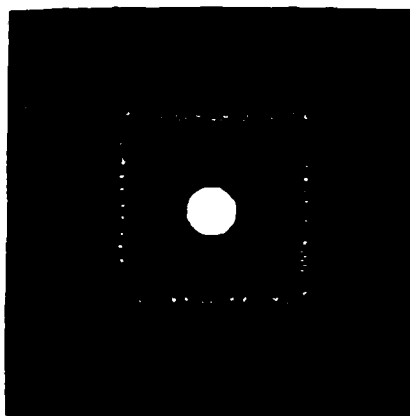


Fig. 2.4. Motion Detection by the PVS.

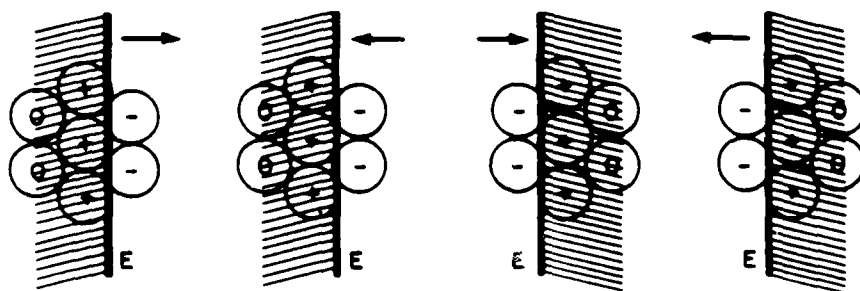
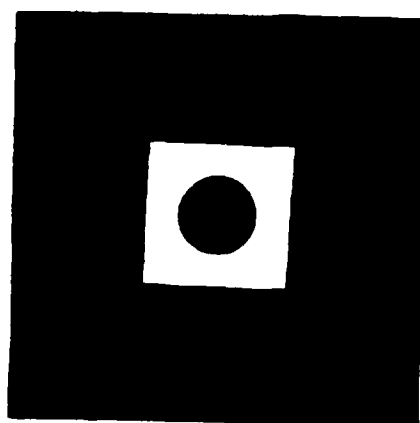
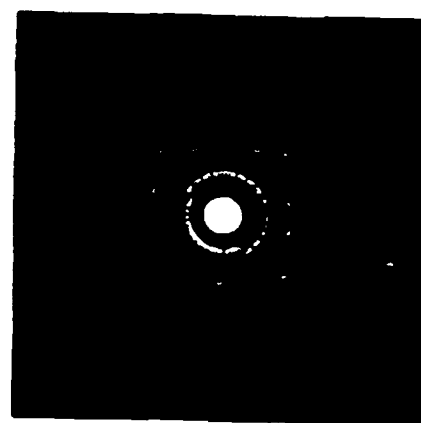


Fig. 2.5. Short-range Motion Detection.



(a) A circle inside a square



(b) The detection motion of a circle and a square

Fig. 2.6. Motion Detection by the New Sensor.

- e) Combining the directions of local motion along the connected zero-crossings by vector analysis, the complete motion is derived. The results were all satisfactory. An example is given in Fig. 2.6. Note that the examples are presented in the image plane for easy visualization.

### References

1. C. Hsin, "Edge and Motion Detection Using Peripheral Human Visual Properties," Master of Science Thesis, University of Virginia, May 1986 .
2. S. Levialdi, "Edge Extraction Techniques," pp. 117-144, in *Fundamentals in Computer Vision*, ed. O. D. Faugeras.
3. R. Gonzalez and P. Wintz, *Digital Image Processing*, Addison-Wesley, Reading, Maine, 1977, pp. 13-16.
4. R. H. Wilson and R. J. Bergen, "A Four Mechanism Model for Threshold Spatial Vision," *Vis. Res.*, vol. 19, pp. 19-32, 1979.

### 3. LONG RANGE QUALITATIVE MOTION DETECTION ALGORITHM

A qualitative algorithm for long-range motion detection has been developed based upon heuristic rules. It has been tested with both synthetic and real images and it has produced correct results in all instances. The assumptions necessary for application of the algorithm are that the image has been segmented and binarized and that a binary edge is available in the computation plane.

Four cases were considered which can be combined to produce more complex scenes:

- a) Objects enclosing the optical axis.
- b) Objects "enclosing the optical axis slightly." It is necessary to make this distinction between cases (a) and (c) due to the discrete nature of the process. For a continuous edge, only (a) and (c) are necessary.
- c) Objects not enclosing the optical axis.
- d) More than one object, of any type, in the field of view.

For objects enclosing the optical axis, the qualitative algorithm is based on having the difference image between the perturbed and the original images in the computation plane; these are called "current image (CI)" and "previous image (PI)," respectively.

The qualitative algorithm is as follows:

Eight (8) position counters are required to keep track of relative positions of points in the CI and the PI. These counters are called *cntc1*, *cntc2*, *cntc3*, *cntc4*, and *cntp1*, *cntp2*, *cntp3*, *cntp4* and they determine which points in each of the frames are in which of the four quadrants. Since the polar angle is mapped to the *v*-axis in the complex plane, the four quadrants represent four distinct regions along it. The four regions over the *v*-domain are shown below. They are

$$-\pi/2 < v \leq \pi/2 \quad (i)$$

$$\pi/2 < v \leq 3\pi/2 \quad (ii)$$

$$0 < v \leq \pi \quad (iii)$$

$$\pi < v \leq 2\pi \quad (iv)$$

The first two regions are used to detect horizontal motion, or motion along the *x*-axis, and the other two to detect vertical motion, or motion along the *y*-axis.

Once the number of points in each region has been obtained, the following set of rules decides the direction of motion.

If there are more CI points in **Region-(i)** than there are PI points in **Region-(ii)** and, at the same time, more PI points in **Region-(i)** than there are CI points in **Region-(ii)**, then the motion is in the **+x** direction. In other words, if the above criteria is true, the object has moved to the right of its original position.

In a similar manner, if the number of CI points in **Region-(ii)** is greater than the number of points in **Region-(i)**, and, at the same time, the number of PI points in

**Region-(ii)** is greater than the number of CI points in **Region-(i)**, the motion is in the  $-x$  direction. In other words, the object has shifted to the left of its original position.

An object is selected to be moving up from its original position if there are more CI points in **Region-(iii)** than there are PI points in **Region-(iv)** and, at the same time, more PI points in **Region-(iii)** than there are CI points in **Region-(iv)**.

For downward motion, the number of CI points in **Region-(iv)** has to be greater than the number of PI points in **Region-(iii)** and, simultaneously, the number of PI points in **Region-(iv)** has to be greater than the number of CI points which fall in **Region-(iii)**.

The verbal description of the detection of horizontal and vertical motion can be confusing. A mathematical formulation describes the situation more succinctly. Let  $N_{P_1}$  be the number of PI points in **Region-(i)**. By the same token,  $N_{P_2}$ ,  $N_{C_1}$ ,  $N_{C_2}$  represent the number of PI points in **Region-(ii)**, and the number of CI points in **Region-(i)** and **Region-(ii)** respectively. Then, if

$$N_{C_1} > N_{P_2} \text{ and } N_{P_1} > N_{C_2}$$

motion is in the  $+x$  direction. Using the same convention, if

$$N_{C_2} > N_{P_1} \text{ and } N_{P_2} > N_{C_1}$$

motion is in the  $-x$  direction.

Defining  $N_{P_3}$ ,  $N_{P_4}$ ,  $N_{C_3}$ ,  $N_{C_4}$  to represent the number of PI and CI points in **Region-(iii)** and **Region-(iv)** respectively, if

$$N_{C_3} > N_{P_4} \text{ and } N_{P_3} > N_{C_4}$$

motion is in the  $+y$  or upward direction. Similarly, if

$$N_{C_4} > N_{P_3} \text{ and } N_{P_4} > N_{C_3}$$

motion is in the  $-y$  or downward direction.

### 3.1. Algorithm Testing

The *Qualitative Algorithm* was tested to have performed satisfactorily with both line images, and synthetically generated two dimensional images. One of the major assumptions made was that an edge image was available to the algorithm. In both cases certain aspects of the images were ideal. The chief of these idealizations was that the edges were well defined and, in most case, one pixel width thick. Noise was introduced in the images to check the robustness of the algorithms, but the edges of the objects were still well defined because of the low content of the noise. In other words, in all of the test cases, the signal-to-noise ratio of the information content of the image was extremely high.

The algorithm was then tested with real images, and it was noted that the edges were not so well defined. In order to remedy this situation, a very high contrast background was used. This provided much better edge information. Once the edges were obtained, the motion of the object was computer simulated. Tests with actual physical motion of the objects were not run.

### 3.2. Problems With Real Images

Real images do not display the ideal properties of either line or synthetically generated images. The actual information in the images is quite heavily overlaid by noise and any attempt made to filter out the noise resulted also in loss of information and hence distortion of the image. Another problem encountered was that of the background. In the images used earlier, the background was of a uniform intensity distribution. This, then, made it a completely negligible entity since it had no features which would either add to or reduce the complexity of the preprocessing algorithm. The background plays an important part in finding the difference images which are needed to find the direction of motion with the qualitative algorithm.

The Gradient operator originally used to detect edges was highly susceptible to noise. Hence, a new operator, viz. the *Sobel Operator*, was utilized to perform edge detection. Though much better than the *Gradient Operator* as far as noise sensitivity is concerned, the *Sobel Operator* produces edges which are on average three pixels wide. Since one of the ideal conditions assumed was that the width of the edges was only one pixel, this caused a serious problem in the application of the algorithms. The images thus have to be heavily preprocessed in order for them to be usable with the qualitative algorithm.

### 3.3. Schemes for Preprocessing Images

A number of schemes were attempted to solve the above mentioned problems. The schemes were all extremely time consuming. A discussion of such problems and the schemes used to counter follows.

The first scheme was used to eliminate the background from the images. A reminder here that the qualitative algorithm uses two images, the second taken a time  $\tau$  after the first, and based on the positional information derived from such an operation, predicts the direction of travel. Since the background was common to the two images, it may be wondered why it has to be eliminated separately. The reason is that a difference image of the two frames needs to be formed. If the background is not eliminated, it is difficult to segment the object from the background after the difference has been found. It was thus easier to eliminate the background from the images before computing the difference.

The following sequence was followed. The first step, or filter, enhanced the edges in the images. This was done by convolving the image with a Sobel Operator. The second step was to *binarize* the image. It is easiest to deal with binary images because segmentation is an important aspect of the sequence. The images thus obtained were then stored for use by another filter which subtracted first the background information from both frames and then the two frames from each other.

Though the above successfully eliminated the problem of interference of the background, one problem remained: the thickness of the edges. This becomes a problem in

terms of speed of computation because for an  $n$  point representation of the edge, the total number of operations performed by the algorithm is of the order of  $n^2$ .

Instead of using edge thinning to solve this problem, another less time consuming scheme was attempted and has provided quite good results. This scheme introduces two extra steps in addition to the sequence mentioned above. After *binarizing* the image, a second edge enhancement operation is performed on the image, this time using a *gradient operator*. The gradient operator produces edges which are two pixel wide, thus reducing the total number of pixels representing the edge.

After the image has been preprocessed, the two images are subtracted from each other. An *average position of difference* is calculated and the edges are then recorded with respect to it. The *average position of difference* is nothing more than a point represented by the average *x-coordinate* and the average *y-coordinate* over the total number of points of same intensity on the two frames. This allows the qualitative algorithm for centred figures to be used exclusively with all images.

It should be mentioned here that the qualitative algorithm being used differs slightly from the scheme utilized with synthetic images. In the earlier case, an edge representation was required, and the number of current image and previous image points in the difference image were used exclusively to calculate the direction of motion. In the scheme used here, due to the extremely noisy image, it is not feasible to trace a contour. Even though the algorithm used earlier did not require continuous edges--a gap of up to 5 pixels width could be tolerated--it was discovered that it failed to perform well with images obtained from the camera. Further testing is being carried out on this aspect of the algorithm.

One major difference between the present use of the qualitative algorithm and the previous use is the following: When using synthetic or line images, since the images underwent a transformation with a computer program, the number of points which constituted the edges remained constant. Hence the probability of incorrectly selecting a direction of motion was not very high since the number of points in any region depended strictly on the mapping. In other words, since the qualitative algorithm uses the number of points in the four quadrants as its basis for determination, the inequalities in the number of points in the various regions were due to the placement of the same number of points in each frame. Preprocessing affects different frames differently in the case of real images. Thus, the number of physical pixels which constitute the current image may be greater or smaller than the number of pixels which constitute the previous edge image. Fortunately, The difference is negligible. Since the typical number of points is of the order of 10,000, a difference of a 100 pixels or so does not constitute a serious error.

### 3.4. Examples and Further Discussion

In the research conducted on real images it was found that the above aspect of the qualitative algorithm was unimportant. The new algorithms were tested with 3 real images. The test objects used were model automobiles.

The results were very encouraging. It was found that even though the pre-processed images contained all the problems mentioned above, the qualitative algorithm correctly predicted the direction of motion in each case. The test results are shown below.

In Fig. 3.1, a car is shown in its initial and moved positions. The movement is to the left and, since the car rests on a slight incline, down. The images are then preprocessed in the sequence described earlier. A second filter is then used to subtract the background from the images and then the images from each other. The first step is shown in Fig. 3.2, and the difference image shown in Fig. 3.3. Note the straight line in Fig. 3.3 dividing the image into two halves. It is in reference to this line that the position of the objects is calculated. The equation of this line is given by:

$$x = \text{average } x\text{-coordinate}$$

As noted earlier, both the x-coordinate and the y-coordinate of the average position are needed. Only one has been shown here for sake of simplicity. The position information of the points which constitute the current (CI) and the previous (PI) frames was previously stored so the new set with respect to the average position of difference is easily obtained. This data is then presented to the direction determining program which first performs the complex logarithmic conformal mapping on the set of data points and then decides upon a direction of motion based on where the points fall in the complex plane. Referring back to the qualitative algorithm, and comparing it against Table 3.1, we note that the conditions for motion to the left and down are met. Hence the algorithm decides *down* and *left* as the directions of motion. A second example is shown in Figures 3.4, 3.5 and 3.6. The data is once again tabulated and is shown in Table 3.2. As can be seen from the data and the above relations, the motion of the car is to its right and in a downward direction.

Once again it is emphasized that in the new use of the algorithm the only use we have of the intersection, or difference, image is to locate a central point. The algorithm performs its operation on the two preprocessed images.

**Multiple Objects** In the earlier work with line images, it was shown that the qualitative algorithm could handle more than one object in the field of view. The objects were not bound to each other in any sense. In other words, the different objects could undergo different transformations and the qualitative algorithm could detect the objects and then

---

Region	Number of Points	
	Previous Frame	Current Frame
(i)	1425	2173
(ii)	3181	2433
(iii)	1299	3307
(iv)	1526	3080

**Table 3.1** Motion to the left and down

---

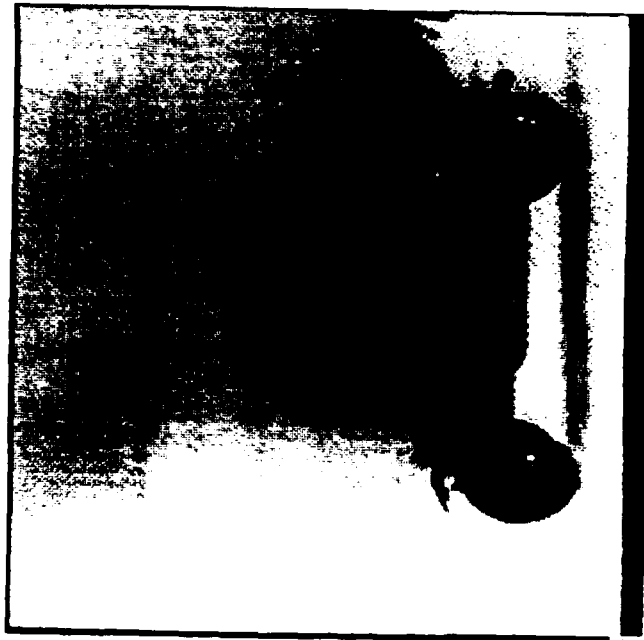
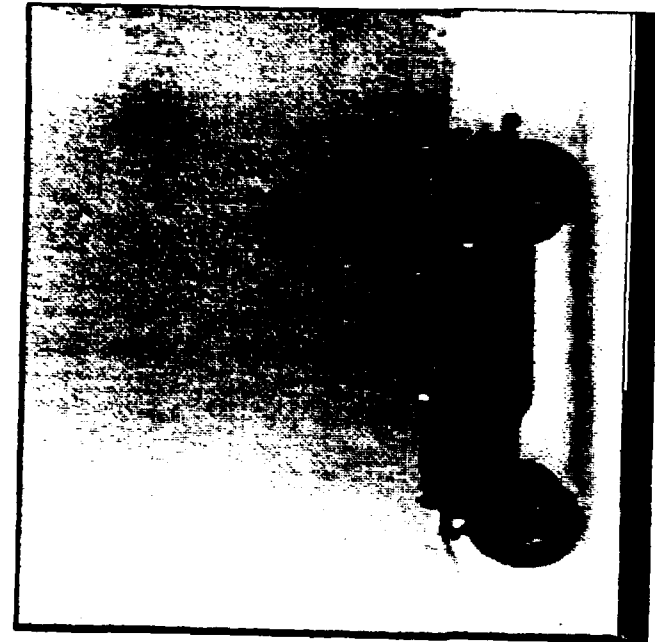


Figure 3.1 Car in initial and moved position.

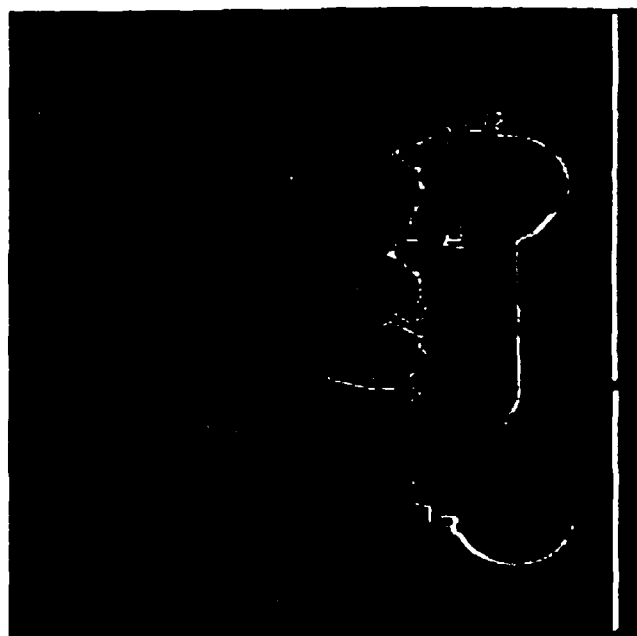
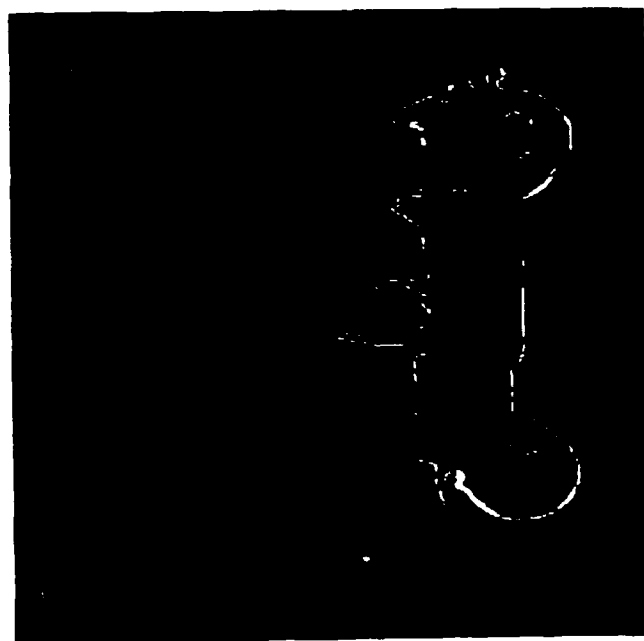


Figure 3.2 Car in initial and moved position after preprocessing.

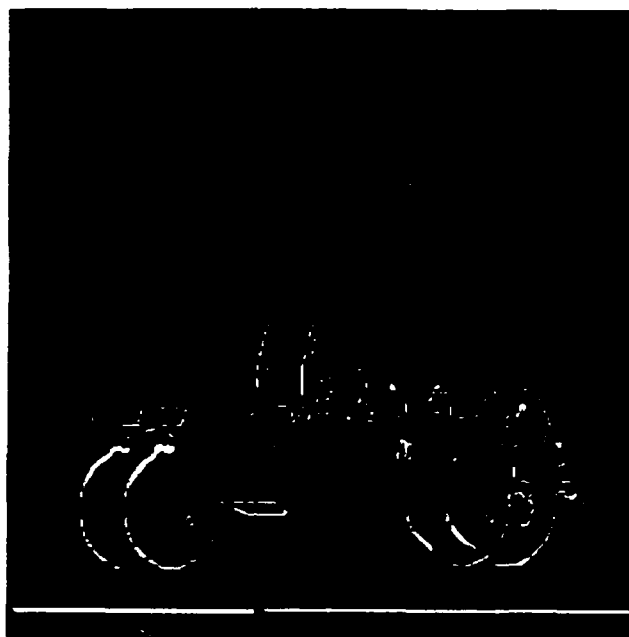


Figure 3.3 Difference Image.

perform the analysis on them. In order to do so, the algorithm depended upon sharply defined edges. The way it found the number of objects in the field of view was to find as many starting points as it could by using an algorithm and then finding connected areas. The limitation of this method was obviously that it could not detect objects whose edges crossed each other.

With real images and lack of clearly defined edges, this does not work very well. At present work is being carried out to make the initial algorithm more robust so that it does not require as well defined edges as it did before. Work is also being carried out to use the motion of the objects to segment them into various shapes. This, of course, presupposes rigidity of the objects. Non-rigid motion is a further topic of consideration. In a sense, the non-uniform logarithmic mapping is nonrigid motion because since the shape of the objects is not preserved, it does not matter so much what the initial orientation is.

At present, work is also being done to develop algorithms to derive depth information from the motion of the objects in a plane. For this purpose, a scheme utilizing the ego-motion of the object is being used. It has been used by various researchers as long as the line of sight and the direction of motion are parallel, or superimposed on each other, there is quite an easy method to derive such information. It is when the two at an angle to each other that problems arise. It is in this area that we are working.

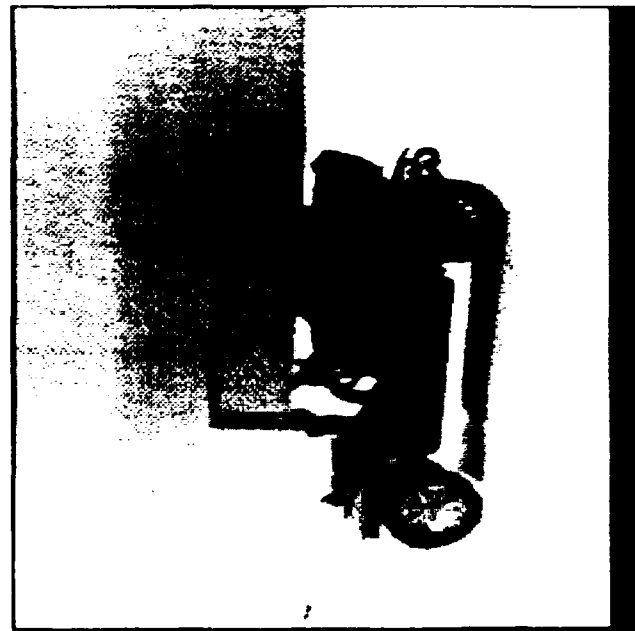
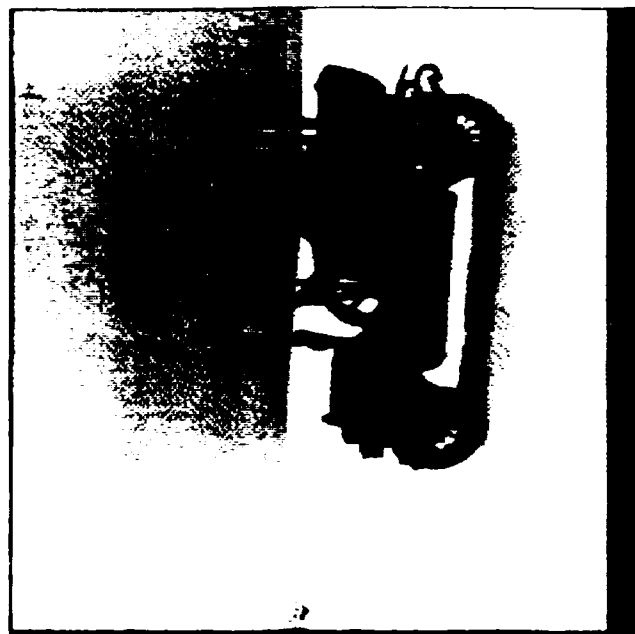


Figure 3.4 Car in initial and moved positions

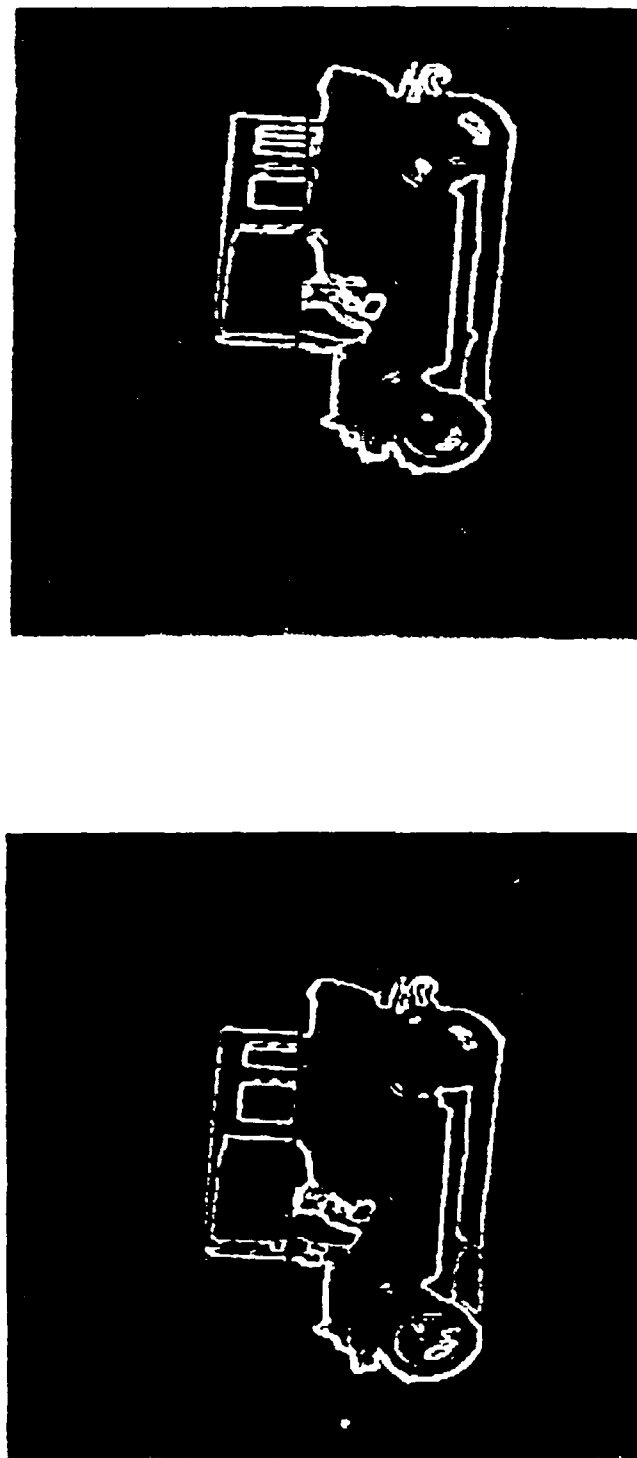


Figure 3.5 Image after preprocessing.

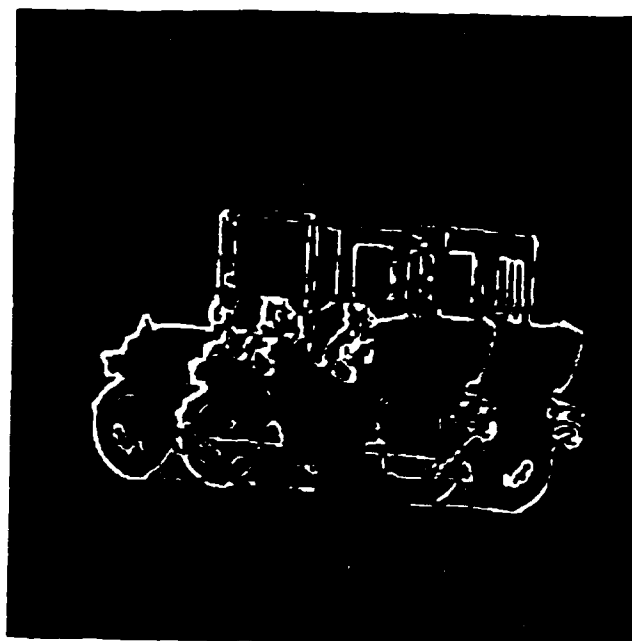


Figure 3.6 Difference Image.

---

---

Region	Number of Points	
	Previous Frame	Current Frame
(i)	1298	1062
(ii)	965	1175
(iii)	337	1885
(iv)	1926	352

---

Table 3.2. Motion to the right and down

---

## References

1. Gilad Adiv, "Determining the Three-Dimensional Motion and Structure from Optical Flow Generated by Several Moving Objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, July 1985, pp. 384-401.
2. B. F. Buxton and H. Buxton, "Computation of Optic Flow from the Motion of Edge Features in the Image Sequences," *Image & Vision Comput.*, vol 2, May 1984, pp. 59-75.
3. R. Jain, "Extraction of Motion Information from Peripheral Processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-3, No 5, September 1981, pp. 489-503.
4. Ramesh Jain, Sandra L. Bartlett and Nancy O'Brien, "Motion Stereo Using Ego-Motion Complex Logarithmic Mapping," *IEEE Proceedings*, 1986, pp. 188-193.
5. Ramesh C. Jain, "Segmentation of Frame Sequences Obtained by a Moving Observer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, September 1984, pp. 624-629.
6. K. Prazdny, "Determining the Instantaneous Direction of Motion from Optical Flow Generated by a Curvilinearly Moving Observer," *Computer Graphics and Image Processing*, vol. 17, November 1981, pp. 238-48.
7. K. Prazdny, "On the Information in Optical Flow," *Computer Vision, Graphics and Image Processing*, vol. 22, May 1983, pp. 239-59.

## 4. A FAST ALGORITHM FOR MOTION PREDICTION

### 4.1 Introduction

A sensor for machine vision with biological visual features has been studied by several researchers [1,2,3,4] based on Schultze's model of the retina [5]. According to this model, the light sensitive array of elements equivalent to the retina, called the "image plane" has receptive fields, or pixels, of increasing size towards the periphery, except for a small region at the center, the fovea, at which pixels are of equal size and uniform distribution. The fovea is the region of highest visual acuity. Outside the fovea, the pixels are distributed in rings whose radius increases in size exponentially. Each ring contains the same number of pixels, all of the same size in a given ring. This pixel configuration in image plane is mapped to the "computation plane", equivalent to the visual cortex by a logarithmic conformal transformation. Figure 4.1 illustrates the sensor configuration both in image plane and computation plane. The transformation holds for all points outside the fovea and it is easy to see that objects scaled or rotated about the optical axis in image plane remain invariant in shape in computation plane, with shifts along the  $u$  and  $v$  axis, respectively, for scaling and rotation. These properties represent important advantages for image processing [6,7]. Examples of these properties can be found in [4].

When objects are translated in the image plane, however, the transformation produces distorted images in computation plane for which it is extremely difficult to perform motion prediction because, in effect, a simple translation in image plane corresponds to nonrigid motion in computation plane.

The receptive fields in a biological visual sensor (BVS) are formed by grouping large numbers of elementary sensors (cones and rods). There seems to be a one to one correspondence between receptive fields and elementary sensors in the fovea region [5,8]. Outside the fovea, translation on the image plane can be analyzed in a simpler way if elementary sensors are grouped to form essentially a rectangular grid. Depending on the visual tasks to be performed, different areas in the visual cortex are used. The visual cortex in primates consists of approximately ten separate areas which are functionally distinct, but complexly interrelated [8] with functions not yet understood. In a vision system emulating some of the features of biological systems, grouping elementary sensors in a rectangular grid to perform tasks such as tracking, is thus justified. If a single rectangular sensor is used, the non-uniform exponential configuration (or "logspiral") is obtained from the sensor by means of a logic circuit [17].

In this section, an original one dimensional correlation tracker for motion prediction is developed and used in both configurations. Two-D motion prediction is performed in the rectangular tessellation image plane and the rotation and scaling motion is performed in logspiral's computation plane.

### 4.2. The Tracking Algorithm

The most relevant characteristic of the algorithm to be described below is that the motion prediction problem is solved without using correspondence. The point-to-point correspondence (defined as : "what point in the new frame corresponds to a given point

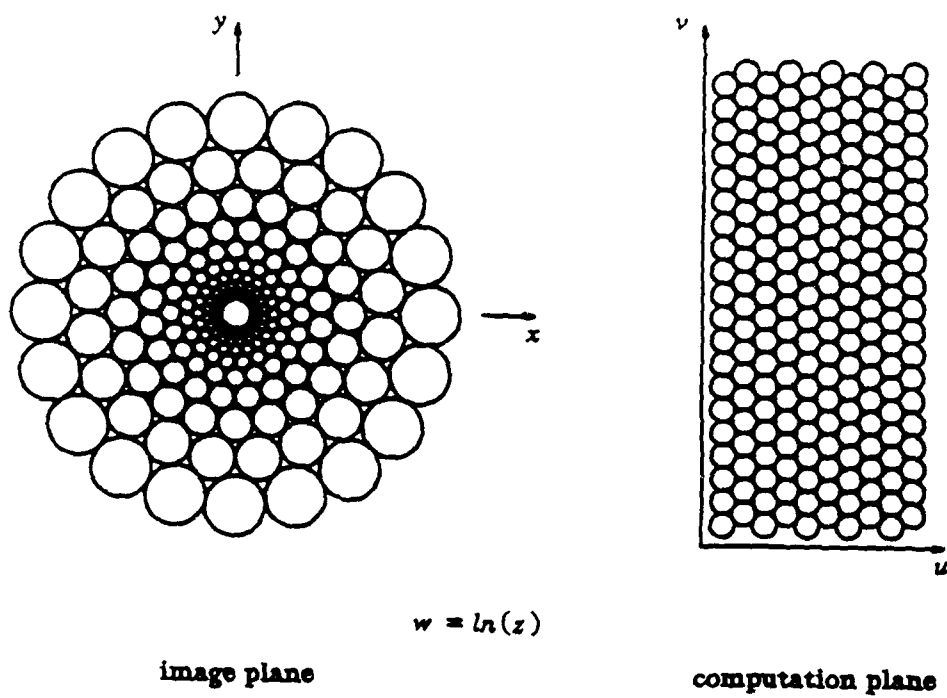


Fig. 4.1. The nonuniform logarithmic spiral sensor, circular elements

in the previous frame.") problem is considered by many researchers to be very difficult, or even unsolvable in practical cases [9,10,11]. Several approaches have been proposed to solve the motion prediction problem without using correspondence [12,13]. The approach proposed in this paper is different from those and produces good experimental results with relatively few computations. Translational motion prediction could be used, in conjunction with the rotation and magnification properties of the logspiral sensor configuration, to predict motion in three dimensions.

Consider a dynamic scene. In general, the intensity of the light reflected by the scene will be a function of location and time,  $I(x,y,t)$ . We can define its gradient in this 3D space,

$$\nabla I = \left[ \frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y} \quad \frac{\partial I}{\partial t} \right]^T \quad (4.1)$$

and its gradient in 2D geometric space,

$$\nabla_x I = \left[ \frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y} \right]^T \quad (4.2)$$

Let  $\hat{u}$  be an arbitrary unit vector in  $(x,y,t)$  space. Then the directional derivative of  $I(x,t)$  in the direction of  $\hat{u}$  is

$$\frac{dI}{ds} = \hat{u} \cdot \nabla I \quad (4.3)$$

where

$$ds = dx\hat{i} + dy\hat{j} + dt\hat{k}$$

For uniform illumination, changes in intensity at a point are due to object motion (assume that other disturbances are inhibited). If the intensity of a point in the object does not change with respect to  $s$  (in other words, an infinitesimal spatial displacement corresponds to a change  $dt$  in time between consecutive frames), then  $\frac{dI}{ds} = 0$ . Let the unit vector  $\hat{u}$  be given by

$$\hat{u} = c(\vec{v}(x,t), 1) \quad (4.4)$$

where  $c = \frac{1}{(|v|^2 + 1)^{1/2}}$  and  $\vec{v}(x,t)$  is the point velocity. Then, combining (4.3) and (4.4) we have

$$\vec{v} \cdot \nabla_x I + \frac{\partial I}{\partial t} = 0 \quad (4.5)$$

Eq. (4.5) is known as the constraint equation relating the spatial gradient to the temporal

derivative for a moving object [14]. Let  $\tau$  be the time between two successive frames,  $\tau = t_2 - t_1$ . We then have

$$\begin{aligned}\Delta(\underline{x}, t) &= \tau \vec{v}(\underline{x}, t) \\ &= (\Delta x \ \Delta y)^T\end{aligned}\quad (4.6)$$

For

$$\frac{\partial I}{\partial t} \approx \frac{I(\underline{x}, t) - I(\underline{x}, t - \tau)}{\tau}\quad (4.7)$$

The approximation of (4.7) is accurate as long as the object motion is relatively small. Using finite increments in (4.5) and replacing  $\vec{v}(\underline{x}, t)$  from (4.6) and  $\frac{\partial I}{\partial t}$  from (4.7),

$$\frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + I(\underline{x}, t) - I(\underline{x}, t - \tau) = 0$$

or

$$I(\underline{x}, t_1) = I(\underline{x}, t_2) + \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y\quad (4.8)$$

In order to deal with large object motion, a recursive process is essential. Thus, to obtain an iterative algorithm, we proceed as follows:

Let's consider only one-dimensional motion for the moment. Rewrite (4.8) as

$$I(x_0, t_1) = I(x, t_2) \Big|_{x=x_0} + \frac{\partial I(x, t_2)}{\partial x} \Big|_{x=x_0} \Delta x,\quad (4.9)$$

where

$$\begin{aligned}\{R_j\} &= \text{Sampled} \left[ I(x_0, t_1) \right] \\ \{T_j\} &= \text{Sampled} \left[ I(x, t_2) \right]_{x=x_j}\end{aligned}$$

and  $x_0$  is an initial pixel position in the first frame.  $x_j$  is an arbitrary chosen pixel position in the second frame. Normally, we will choose  $x_j$  to be equal to  $x_0$  unless a priori information about  $x_j$  is provided. Thus, for any pixel and for a discrete case, we have

$$\{R_j\} = \{T_j\} + \frac{\partial I(x, t_2)}{\partial x} \Big|_{x=x_j} \Delta x,\quad (4.10)$$

where

$$\frac{\partial I(x_j, t_2)}{\partial x_j} \approx T_{j+1} - T_j$$

Now, define the correlation of  $\{T_j\}$  and  $\{R_j\}$  as

$$C = \sum_{j=1}^m R_j T_j \quad (4.11)$$

Combine (4.10) and (4.11) together and for  $\Delta x_i = \Delta x_j$  for all  $j$ , yield

$$\Delta x_i = \frac{\sum_{j=1}^m R_j^2 - \sum_{j=1}^m R_j T_j}{\sum_{j=1}^m R_j \frac{\partial I}{\partial x_j}} \quad (4.12)$$

Similarly, for  $\Delta y_i$  expression, we have

$$\Delta y_i = \frac{\sum_{j=1}^m R_j^2 - \sum_{j=1}^m R_j T_j}{\sum_{j=1}^m R_j \frac{\partial I}{\partial y_j}} \quad (4.13)$$

where

$$\begin{aligned} \{R_j\} &= \text{Sampled} \left[ I(y_0, t_1) \right] \\ \{T_j\} &= \text{Sampled} \left[ I(y, t_2) \Big|_{y=y_j} \right] \end{aligned}$$

In other words,  $\Delta y_i$  is estimated in a column direction.  $y_0$  and  $y_j$  are as before. To start the algorithm above, we let

$$\begin{aligned} \Delta x_0 &= 0 \\ \Delta y_0 &= 0 \\ x_i &= x_{i-1} + \Delta x_{i-1} \\ y_i &= y_{i-1} + \Delta y_{i-1} \end{aligned} \quad (4.14)$$

and  $R$  is evaluated for the next iteration as

$$\{R_j\} = \text{Sampled} \left[ I(x, y, t_1) \Big|_{x=x_j} \right] \quad (4.15)$$

The iteration process stops when the following conditions occur

$$\begin{aligned}x_i &= x_{i-1} \\ y_i &= y_{i-1}\end{aligned}\tag{4.16}$$

or

$$\begin{aligned}\Delta x_i &= 0. \\ \Delta y_i &= 0.\end{aligned}\tag{4.17}$$

After stopping the iteration process, the final estimated expressions are

$$\begin{aligned}\Delta x &= \sum_{i=1}^n \Delta x_i \\ \Delta y &= \sum_{i=1}^n \Delta y_i\end{aligned}\tag{4.18}$$

where  $n$  is the number of iterations. Equation (4.18) gives us the estimates within a fraction of pixel accuracy. The final estimates can also be obtained equivalently from the following equations.

$$\begin{aligned}\Delta x &= x_i - x_0 \\ \Delta y &= y_i - y_0\end{aligned}\tag{4.19}$$

These expressions, however, can only give us the estimates within an integer value.

The algorithm developed above is a spatio-temporal type algorithm (gradient algorithm) which is very attractive as far as hardware implementation is concerned [15,16]. The algorithm utilizes only a point-by-point basis (i.e.; a single-degree-of-freedom correlation) as opposed to conventional correlation. Thus, computation time is significantly reduced. But, perhaps, the most important feature is its ease of implementation in a parallel fashion. Horizontal and vertical perturbations can be computed independently even if the object experiences 2-D motion. The reason for this is that (4.12) and (4.13) are based on the individual row or column. Thus, rows and columns can be processed simultaneously. Thus, for example, an image of size  $64 \times 64$  pixels will need  $2 \times 64$  cpu seconds to compute the perturbations, whereas for the parallel processors with 128 processors, the time required is  $t$  seconds, where  $t$  is the processing time to compute the estimate in a single row. In addition, the algorithm requires computation of gradients. This, in general, causes a serious problem when the image is severely degraded by noise. In the real image simulation, we will demonstrate empirically that the algorithm above possesses good noise immunity and the computation of the gradients does not affect the estimates severely.

#### 4.2.1. 2D Motion (a general case)

The correlation tracker has been derived considering motion in the  $x$  and the  $y$  directions independently, and although it can be applied to motion in the  $x$  and  $y$  directions, in general it will not produce good results for motion in both directions simultaneously. The problem for general  $x/y$  translation is that corresponding rows (and columns) in frames  $i$  and  $i+1$  within the moving area will be shifted with respect to each other. For example, if  $\Delta x = 3$  pixels and  $\Delta y = 5$  pixels, the object has moved right by 3 pixels and up by 5 pixels and rows in frame  $i+1$  will be at location  $k+5$  with respect to corresponding rows in frame  $i$  (where  $k$  is row number). Hence, if a search is performed for each row in both the positive and the negative directions to find the best matching row in the second frame, this problem can be solved. The question remains: how many rows (and columns) must comprise the search area? This, of course, will depend on how large a motion will occur between frames which, in turn, depends on sampling rate and object speed. A reasonable figure is to allow for displacements of at most ten percent of the maximum object dimension in pixels, in both the  $x$  and the  $y$  directions. The number of search rows will then be twice the maximum object dimension plus one, in pixels units.

The row under investigation is correlated with all the rows in the search region and a pixel by pixel estimate is performed. The row which produces the most consistent estimate is likely to be the matching one. This can be best illustrated by Fig. 4.2, where we search 3 rows above and 3 rows below the  $k$ th row in the first frame. If the object, for example, moves up 2 pixels, then the  $(k-2)$ th row in the second frame is perfectly matched to the row  $k$ th in the first frame. That is, the estimate obtained from correlating these two rows using (4.12) is a consistent estimate. Others will produce inconsistent estimates. This will become clearer in the simulation on a Gaussian image. It is also possible that the first iteration, although producing the most consistent result, will not suffice, so the search region is decreased to a fraction of its previous size and a second iteration is performed. After the first iteration, in order to determine which is the row with the most consistent estimation, a histogram of pixel estimates is generated for each row (or column). The histogram with the smallest variation is chosen as the matching one. After this first iteration, the average estimate for that row is chosen (nearest integer) as the first displacement estimate and the reference image (first frame) is moved by that number of pixels in the estimated direction of motion. A new iteration is now performed using (4.12) and the process is repeated. As mentioned above, there may be cases in which the original best matching row is not the same for the second iteration. When the displacement estimate is zero or a fraction of a row less than one half, the iteration process is stopped. All estimated values are then added to determine the total displacement estimate.

#### 4.2.2. Simulation with a Gaussian Grey Level Image

The algorithm was tested with a bi-valued Gaussian function of the form

$$p(x,y) = 10^7 [P_1 p_1(x,y) + P_2 p_2(x,y)]$$

with

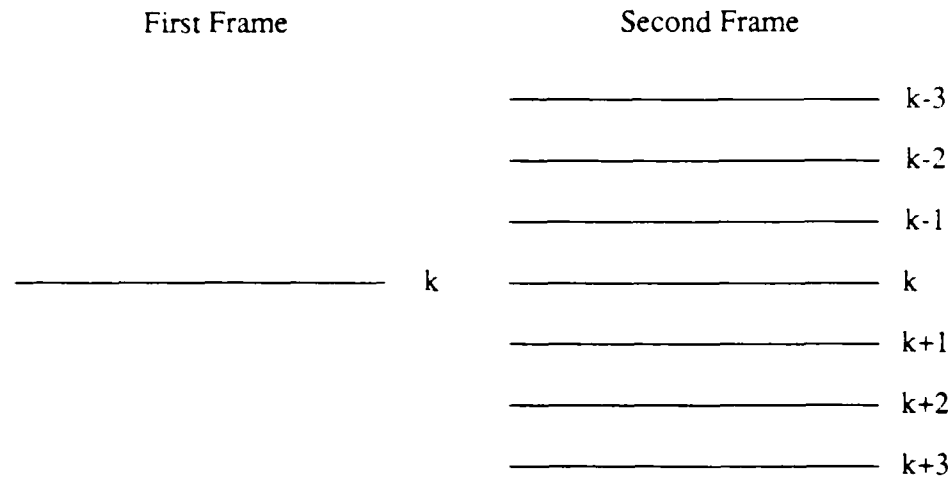


Figure 4.2 Row search mechanism

---

$$p_i = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\delta_i^2}} \exp \left\{ -\frac{1}{(1-\delta_i^2)} \left[ \left( \frac{x-m_x}{\sigma_x} \right)^2 - \frac{2\delta_i(x-m_x)(y-m_y)}{\sigma_x\sigma_y} + \left( \frac{y-m_y}{\sigma_y} \right)^2 \right] \right\}$$

The array size is 64×64,  $-20 \leq x \leq 43$ ,  $-25 \leq y \leq 38$ . Figure 4.3 shows the grey level image.

The synthetic image was displaced by  $\Delta x=3$ ,  $\Delta y=3$  and a row by row correlation was performed using (4.12) with a search area of nine rows. The algorithm is applied only to the minimum rectangle enclosing the object (tracking window) to be tracked, hence we are assuming that segmentation has already been performed offline. This needs to be done only at the start of the operation. In other words, a priori information about target characteristics such as its size and its location with respect to the reference frame need to be known. The average displacement estimate for rows in the region containing the image when correlated with rows displaced from  $k-4$  to  $k+4$  is shown in Table 4.1. The corresponding histograms, i.e., the number of occurrences vs estimated displacement for rows  $k$  and  $k+i$ ,  $i=-4$  to  $4$ , are shown in Fig. 4.4. Comparing the table with the histogram, a best match is obtained for a row shift equal to three, with a  $\Delta x \approx 2$ . In this case, the matching rows in the second frame produce consistent estimates of  $\Delta x$ . Other row shifts give estimates of  $\Delta x$  which vary widely in both magnitude and direction. The original image is now shifted by 2 pixels to the right and a new iteration is performed. A similar

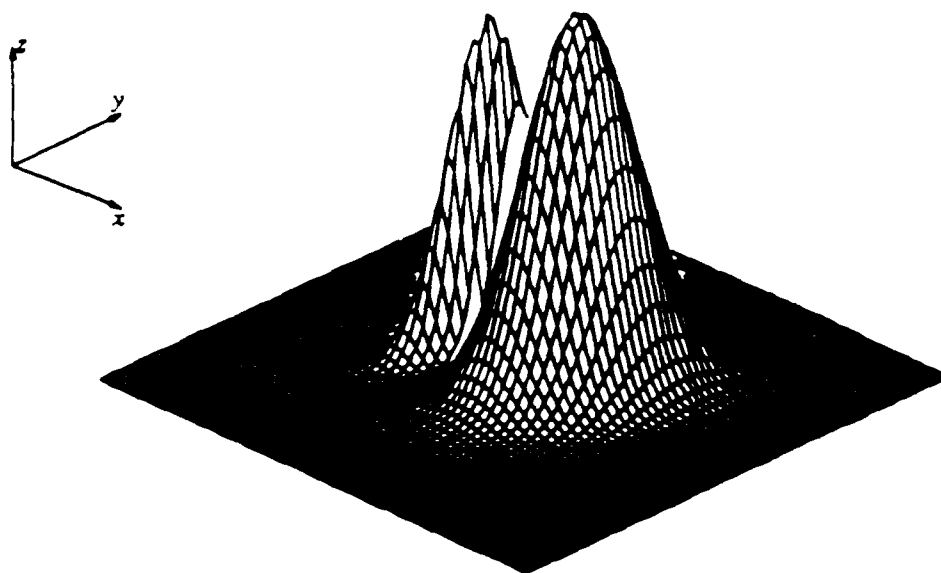


Fig. 4.3. Mixed density Gaussian grey level image

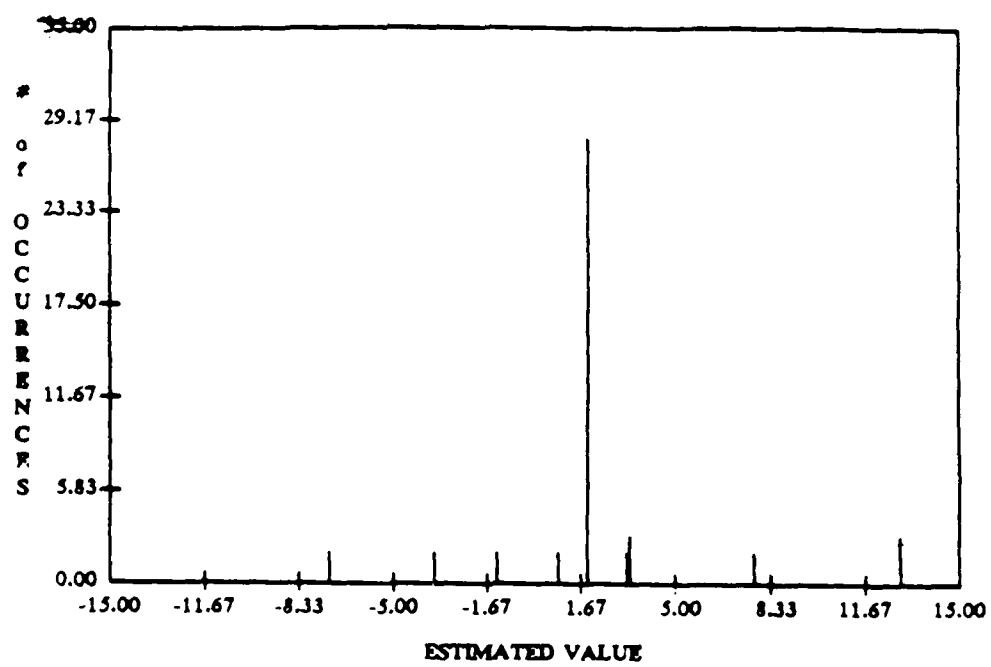


Fig. 4.4. Histogram for correlation of rows  $k$  and  $k+1$

procedure is applied to columns alternatively (or in the case of parallel processing, simultaneously). This second iteration produces a displacement estimate of  $\Delta x \approx 1.00$ . The process is repeated and the next iteration results in  $\Delta x = 0.00$ , thus the iterative procedure is ended and the estimated displacement in the  $x$  direction, according to (4.18) is  $\Delta x = 2.93930$  units or, in integer pixels, 3 pixels.

The procedure is applied simultaneously to rows and columns, i.e., after the first row iteration, the first column iteration is performed, etc. Table 4.II indicates the results for  $x$  and  $y$

Table 4.I Row correlation for search area, a Gaussian image

row k	Correlation With Row								
	k-4	k-3	k-2	k-1	k	k+1	k+2	k+3	k+4
31	-1229.91	-330.37	-119.03	-56.27	-42.52	170.40	5.49	1.90	2.15
32	-394.03	-126.39	-50.32	-26.25	-19.56	70.79	2.61	1.90	3.29
33	-132.70	-46.85	-20.10	-9.92	-5.68	3.57	0.23	1.90	4.66
34	-43.65	-16.06	-6.02	-1.17	3.67	-23.65	-1.71	1.90	6.34
35	-13.21	-3.52	0.89	4.42	12.47	-23.96	-3.05	1.90	8.91
36	-1.16	3.67	7.11	12.89	67.83	-13.42	-2.98	1.90	12.48
37	301.63	-188.25	-51.52	-20.50	-8.72	-2.67	1.11	1.86	-17.34
38	-5.53	-4.52	-2.84	-0.35	3.32	7.80	9.38	1.83	-7.96
39	-1.27	0.37	3.17	8.04	15.68	21.39	13.88	1.83	-5.14
40	2.71	6.10	12.52	24.18	37.39	31.69	13.52	1.83	-3.77
41	9.26	17.43	33.79	58.13	59.03	31.46	11.55	1.83	-2.67
42	22.41	43.81	83.73	102.40	60.65	26.03	9.47	1.83	-1.57
43	53.39	113.37	174.90	112.50	48.33	19.82	7.31	1.83	-0.43
44	145.77	321.73	232.65	87.07	34.20	13.76	5.32	1.83	0.79
45	925.65	1216.69	191.13	57.86	21.44	8.49	3.42	1.83	2.04
46	-443.61	12935.43	115.41	31.20	10.58	3.66	1.62	1.83	3.35
47	-171.40	-1654.64	51.91	10.39	1.28	-0.68	-0.09	1.83	4.78
48	-68.31	-712.92	1.15	-7.25	-6.84	-4.70	-1.76	1.83	6.25
49	-0.09	692.00	-38.54	-21.48	-13.90	-8.27	-3.32	1.83	7.76
50	47.63	880.75	-70.04	-33.86	-19.93	-11.60	-4.82	1.83	9.46
51	84.12	7170.58	-91.99	-42.33	-24.92	-14.47	-6.17	1.83	11.11
52	107.24	1975.81	-111.11	-51.17	-29.73	-17.15	-7.66	1.83	12.92
53	123.95	8112.47	-124.42	-57.23	-33.06	-19.67	-8.82	1.83	14.73
54	132.00	1557.10	-147.05	-63.92	-37.97	-22.17	-10.26	1.83	16.50
55	137.75	1363.57	-151.96	-69.80	-40.18	-24.09	-11.11	1.84	19.00
56	140.48	1465.12	-169.98	-73.99	-43.33	-25.88	-12.37	1.89	20.67
57	190.06	-1307.00	-129.74	-66.07	-39.85	-24.97	-12.15	1.91	22.78
58	137.04	817.50	-210.38	-79.37	-52.29	-29.95	-15.60	2.10	22.71
59	131.95	1930.00	-139.89	-64.17	-42.60	-29.71	-13.40	3.00	30.50

estimates.

Due to the low resolution used for the sensor in this example, there are situations in which the estimated displacement contains significant error. Table 4.III summarizes the results of several displacements computed by either (4.18) or (4.19). Notice that, in general, the results obtained from (4.19) are better, although when they are rounded up to an integer number of pixels (as for example in the last case,  $\Delta x = -4$ ,  $\Delta y = -1$ ) there can still be an error. This, however, lies within the quantization error of  $\pm 1/2$  pixel and, if the motion is the same but the sensor resolution is increased, the results are better (see [17] for a real image 2D translation simulation).

### 4.3. TRANSLATION, ROTATION, AND SCALING

To track 3D motion, the two planes (rectangular and logspiral's computation) will be used simultaneously. During a processing period, information from each plane will be passed back and forth. This is important due to the fact that if a logspiral sensor (image plane) is used, translation in image plane produces a distorted image in computation plane. On the other hand, rotation about and translation along the optical axis are dealt with easily using the logspiral sensor (computation plane), but are difficult to deal with in the rectangular (image plane) sensor. Thus, to accomplish the task, the logspiral sensor's computation plane and the rectangular sensor's image plane will have to interact. There is useful information in both of them, depending on the type of motion, that can be used to optimize the process. Notice that rotation and scaling considered here are both with respect to the optical axis. Thus, no rotation of the image plane on other axes is allowed.

Let's consider a combination of translation and rotation for the moment. If we can arrange a group of processors in such a way that each of them is responsible for a different object orientation, then it may be possible to estimate  $x$  and  $y$  perturbations plus rotation with respect to the optical axis simultaneously. This arrangement is analogous to a model proposed for the visual

Table 4.II A summary of horizontal and vertical perturbations for a Gaussian image

Displacement $\Delta x = 3.0$ , $\Delta y = 3.0$		
Iteration	$\Delta x_j$	$\Delta y_j$
1	1.93930	1.99919
2	1.00000	1.00000
3	0.00000	0.00000
$\Delta x = \sum_{j=1}^3 \Delta x_j = 2.93930$		
$\Delta y = \sum_{j=1}^3 \Delta y_j = 2.99919$		

Table 4.III More simulations for a Gaussian image

Displacement	Iteration	$\Delta x_j$	$\Delta y_j$	$\Sigma \Delta x_j$	$\Sigma \Delta y_j$	$x_i - x_0$	$y_i - y_0$
$\Delta x = 4.0$ $\Delta y = 1.0$	1	2.55447	1.00000	3.55447	1.00000	4.00000	1.00000
	2	1.00000	0.00000				
	3	0.00000	-				
$\Delta x = -4.0$ $\Delta y = 3.0$	1	-2.21238	1.99919	-3.56421	2.99919	-3.00000	3.00000
	2	-0.939303	1.00000				
	3	-0.412527	0.00000				
$\Delta x = -4.0$ $\Delta y = 3.0$	1	-2.21238	1.99919	-3.56421	2.99919	-3.00000	3.00000
	2	-0.93930	1.0000				
	-	-0.412527	0.0000				
$\Delta x = -4.0$ $\Delta y = -1.0$	1	-2.62900	-0.439969	-3.56421	-0.439969	-4.00000	0.00000
	2	-0.939303	-				
	3	-0.412527	-				

cortex [18]. That is, we will use the tracking windows with different orientations. These windows will be used by each processor which in turn computes the translational parameters. We then look at the most consistent estimates. In other words, we will use the same procedure as before except that we now apply the algorithm to various orientations instead of one. The consistent estimates should occur when the reference and the target have the same orientation as well as when its rows and its columns are perfectly matched. This technique can be extended to case of translation, rotation, and scaling. All we have to do is, at a given orientation, is to generate various tracking windows each with a different scaling factor. Thus, a processor having a tracking window which best matches the target both in orientation and scaling will give the highest response. In other words, it gives the most consistent estimates. Other processors will also respond but their estimates will not be as consistent as the one having the best match tracking window. This is where a similarity between this technique and the model in [18] appears. Notice that this is **not the same as the conventional correlation**. What we intend to do here is to employ a one-dimensional correlation algorithm and implement it with a parallel architecture.

The following strategy is proposed for the two planes interaction.

### I. Logspiral Plane (Computation Plane)

1. Apply the algorithm previously developed to both angle (column) and radius (row) direction.
  - a). If the estimates are not consistent within some degree, then some kind of motion must have occurred. Signal the rectangular plane to start operation.
  - b). If the estimates are consistent within some degree, then the object experiences only rotation or scaling or both. Thus, there is no 2D translation involved. No operation is necessary in the rectangular plane.
  - c). If the estimates are zero, then nothing has happened. That is, no motion whatsoever has taken place.
2. Receive a start signal from the rectangular plane. Then, apply the same procedure as used in step 1.
3. Repeat step 2 until the algorithm converges.

### II. Rectangular Plane (Image Plane)

1. Receive a start signal from the logspiral plane. Then, take the following actions:
  - a). Calculate  $\Delta x$  and  $\Delta y$  using the same procedure as before using some tracking windows at a relatively low "resolution" or using the windows at the orientation and scaling suggested by the computation plane to narrow down the region of operation.
  - b). Look at the most consistent estimates which are the best estimates at this point.
  - c). Update the original reference according to the best  $\Delta x$  and  $\Delta y$ .
  - d). Remap the reference via logarithmic transformation.
  - e). Signal the logspiral plane to start the operation.
2. Receive a start signal from the logspiral plane. Then, take the following actions:
  - a). Generate a finer window "resolution" according to where the most consistent estimate (step 1) occurs or according to the angle and scaling information from the logspiral plane, again, to narrow down the region of operation.
  - b). Calculate  $\Delta x$  and  $\Delta y$  using the same procedures as in step 1, but this time use the windows generated in a).
  - c). Repeat part b through e in step 1.
3. Repeat step 2 until the algorithm converges.

The above strategy will be used during a real image simulation. Notice that in step 1, the computation plane may or may not give any information about rotation and scaling. This depends on how inconsistent the estimates in the computation plane are. We shall pursue this matter in the simulation section.

#### 4.3.1. Simulations with a Real Image

The technique proposed above was tested with a real image taken from a CCD camera. The object is a picture of the deck of a submarine Fig. 4.5. The background is dark with several lighter wide spots, in various degrees of intensity. The image mounted on a metal plate, is placed on top of a vertical steel rod, which can be rotated manually, in which angles can be read accurately by means of a vernier, to a fraction of a degree. The camera is placed directly above the object. It is mounted on a steel camera holder which can be moved vertically. The object is placed at an arbitrary position initially. The image is then taken with a resolution of  $512 \times 512$ . This represents our first frame. The second frame is obtained by first manually moving the object in the  $x$  direction by 0.5 cm and in the  $y$  direction by 0.5 cm. The vertical steel rod is then rotated by  $5^\circ$ . Finally, the camera is moved vertically up 5 cm. The image is then taken. Thus, we have the second frame which represents the target, whereas the first one is our reference, of course. Notice that the background is rotated and scaled along with the object for this particular experiment set up.

In order to be able to check whether the algorithm performs satisfactorily, we need to know to what number of pixels in the  $x$  and  $y$  directions and to what  $\Delta\theta$  and  $\kappa$ , the physical motion experienced by the object corresponds. This is, in a sense, a "calibration" procedure. In this way, we can compare the known  $\Delta x$ ,  $\Delta y$ , and  $\Delta\theta$  and  $\kappa$  to the results obtained from the algorithm. This was achieved using standard image processing techniques. The following are all parameters resulting from those procedures.

Background's location (relative to an image coordinate)

Top left corner (96,153)    Top right corner (96,342)  
Bottom left corner (327,153)    Bottom right corner (327,342)

Object's location (relative to an image coordinate)

Top left corner (142,179)    Top right corner (142,305)  
Bottom left corner (298,179)    Bottom right corner (298,305)

Background size  $9 \times 9$  cm =  $189 \times 232$  pixels

Object size  $6 \times 6$  cm =  $126 \times 156$  pixels

The four actual motion parameters are

$\Delta x = 0.5$  cm = +5 pixels  
 $\Delta y = 0.5$  cm = -4 pixels  
 $\Delta\theta = 5.0^\circ = +11^\circ$   
 $\kappa = 5.0$  cm = 0.84 (dimension reduced)

Before applying the algorithm, we will set the tracking window to the size of the object plus 10% on each side. The tracking window will be used exclusively in the rectangular plane. The logspiral's computation plane used consists of arcs of rings configuration with 60 rings, 64



**Fig. 4.5. Real image for simulation**

elements per ring, and an exponential spacing constant of 1.04. We are now ready to apply the algorithm for estimating the perturbations.

First, we apply the algorithm in the computation plane. The results show in the first column of row and column estimates Table 4.IV. As can be seen, the estimates in both columns are not consistent to within some preset threshold. However, they are telling us that the object has become smaller. Furthermore, it rotates by about  $12^\circ (2.14 \times \frac{360}{64})$ . The rectangular plane uses this information by setting its windows conservatively at  $3^\circ, 6^\circ, 9^\circ, 12^\circ$ , and  $15^\circ$  for rotation and .75, .80, .85, .90, .95 for the scaling factor. One could have narrowed the operating area a little more if desired. The results for the first iteration are shown in Table V(a,b), whereas, their corresponding histograms are illustrated in Fig. 4.6(a,b). From the table, the horizontal and vertical perturbations are 2.67 for  $\Delta x$  and -0.906 for  $\Delta y$ . The second iteration then proceeds from this point using the exact same method as before. During this iteration, again, the estimates in the computation plane can be used. Notice that they are getting more consistent and that the angle information is around  $10^\circ$  now. The rectangular plane uses finer resolution both in orientation and in scaling (see Table 4.VI(a,b) and Fig. 4.7(a,b)). From the tables,  $\Delta x$  and  $\Delta y$  equal to 1.83 and -1.94, respectively. During the third and fourth iterations, the estimates are more or less the same (see column 3 and 4 for both row and column estimates, in Table 4.IV). The rectangular plane also processes for two more iterations before it converges. Table 4.VII summarizes the estimates in the rectangular plane. The overall estimates for horizontal and vertical perturbations are 4.565 for  $\Delta x$  and -3.834 for  $\Delta y$ . The rotation and scaling parameters from both planes (they agree to within a small percent error) are approximately  $10^\circ$  and 0.83, respectively.

The results are very satisfactory although they are not exact as predicted. Actually, we do not know exactly what the true perturbations are due to inevitable errors in the translation and scaling displacements, specially. Furthermore, in most tracking applications, the estimates do not need to be exact but they must be accurate to within some percents. These arguments also apply to the iteration processes. That is, the rectangular plane does not have to use its windows from  $3^\circ$  to  $15^\circ$  for rotation and .75 to .95 for the scaling factor. Instead, it could have used finer resolution in the first place. This, of course, depends on how useful the information from the computation is. If the estimates are too inconsistent, the computation plane is better not to tell anything and vice versa. The degree of inconsistency of the estimates in turn depends on how large the motion is. If it is not too large as in this case, then estimates are not quite inconsistent as they ought to be. For a fast sampler, this could well be the case. Notice that the degree of inconsistency of the estimates can be observed from Table 4.IV. As can be seen, as the 2D translation is getting smaller (by updating the reference), the more consistent the estimates will be.

#### 4.4. Speed Comparison

In this section, we compare the speed of one-dimensional correlation and conventional correlation. Although speed comparison with other algorithms will not be made, the analysis may be conducted in a way similar to the one used here. Parallel processing for both algorithms will not be considered. We will use a serial processor for both of them.

##### 4.4.1. Conventional Correlation

As is well known, direct correlation can be used to find target perturbations. This technique requires a tremendous amount of processing time. An advantage of the technique is that a solution is guarantee as long as we are willing to wait. For a large image size ( $n \geq 128$ ), the correlation is carried out in the frequency domain rather than in the spatial domain. The images that need to be correlated are transformed to the frequency domain using the Fast Fourier Transform (FFT). The matrix (complex number) multiplication is then performed on the resultant images. The

Table 4.IV A Real Image with  $\Delta x = 5.0$ ,  $\Delta y = -4.0$ ,  $\Delta \theta = 11.0^\circ$ ,  $\kappa = .84$ 

Computation Plane Estimates							
Row ( $\kappa$ )				Column ( $\Delta \theta$ )			
1	2	3	4	1	2	3	4
-3.0	-4.0	-4.0	-4.0	-	-	-	-
-3.0	-4.0	-5.0	-5.0	-	-	-	-
-3.0	-4.0	-4.0	-4.0	-	-	-	-
-3.0	-4.0	-5.0	-5.0	-	-	-	-
-3.0	-4.0	-4.0	-4.0	-	-	-	-
-5.0	-4.0	-5.0	-5.0	-	-	-	-
-6.0	-5.0	-5.0	-5.0	-	-	-	-
-6.0	-6.0	-5.0	-5.0	-	-	-	-
-6.0	-6.0	-5.0	-5.0	-	-	-	-
-6.0	-6.0	-6.0	-5.0	-	-	-	-
-5.0	-5.0	-5.0	-5.0	-	-	-	-
-5.0	-5.0	-5.0	-4.0	-	-	-	-
-4.5	-6.0	-5.0	-5.0	3.0	2.0	2.0	2.5
-7.0	-5.0	-5.0	-4.0	3.0	2.0	2.0	2.0
-7.0	-5.0	-6.0	-5.0	1.5	1.5	2.0	2.0
-7.0	-6.0	-4.0	-5.0	1.5	1.5	1.5	1.0
-2.33	-2.0	-3.5	-3.5	2.0	2.0	2.0	2.0
-10.0	-4.0	-9.0	-8.0	2.33	2.0	3.5	3.5
-10.0	-5.0	-8.0	-7.0	1.75	1.5	1.75	1.75
-4.0	-6.0	-8.0	-8.0	1.5	2.0	1.75	1.75
-4.0	-6.0	-6.0	-7.0	2.25	1.75	1.75	1.75
-4.0	-6.0	-5.0	-4.0	1.75	7.0	2.0	2.0
-4.0	-9.0	-5.0	-5.0	3.0	-	-	-
-4.0	-9.0	-4.0	-5.0	-	-	-	-
-4.0	-9.0	-5.0	-5.0	-	-	-	-
-4.0	-6.0	-5.0	-5.0	-	-	-	-
-3.0	-6.0	-5.0	-5.0	-	-	-	-
-3.0	-4.0	-5.0	-5.0	-	-	-	-
-3.0	-4.0	-5.0	-4.0	-	-	-	-
-3.0	-4.0	-4.0	-5.0	-	-	-	-
-	-	-	$\kappa \approx -5.05$	-	-	-	$\Delta \theta \approx 2.025$

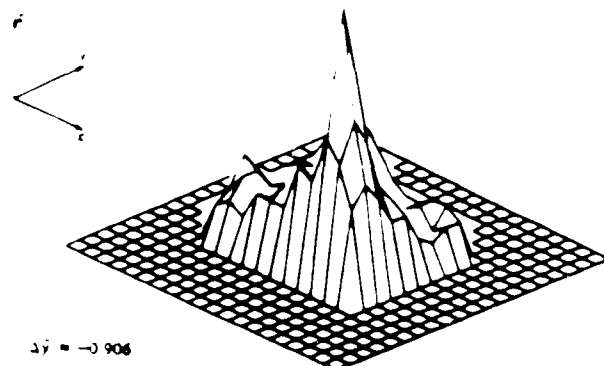
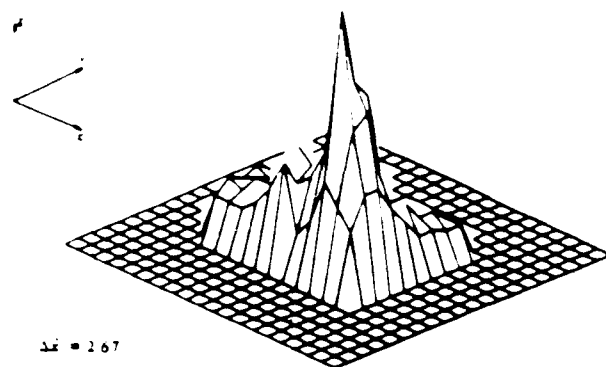


Fig. 4.7(a,b). Histograms of Table 4.V(a,b)

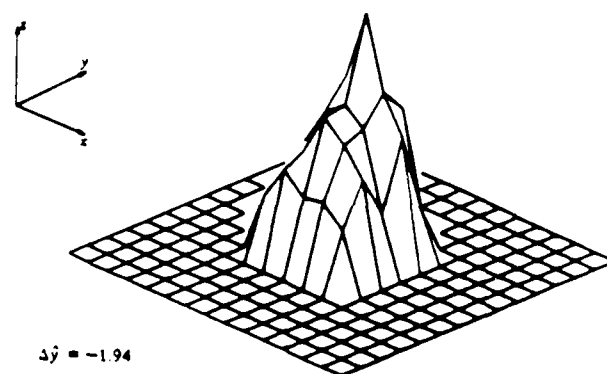
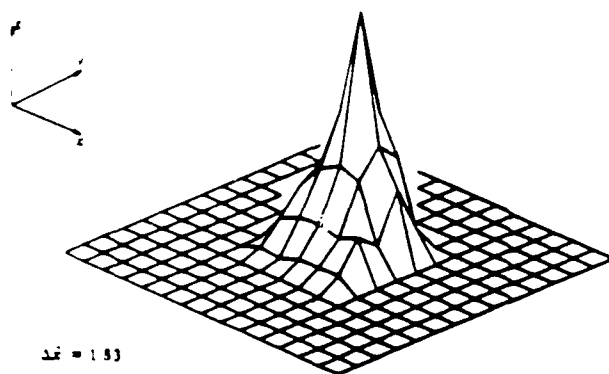


Fig. 4.7(a,b). Histograms of Table 4.VI(a,b)

Table 4.V(a) First Iteration  $\Delta x=5.0, \Delta y=-4.0, \Delta \theta=11.0^\circ, \kappa=.84$ 

Scaling Factor (row estimates)					
Angle	.75	.80	.85	.90	.95
3.0°	-2.56 (7)	4.89 (15)	4.64 (15)	6.58 (17)	8.88 (6)
6.0°	5.19 (10)	2.51 (20)	5.26 (19)	7.41 (20)	9.13 (10)
9.0°	0.042 (17)	2.67 (43)	5.42 (31)	8.18 (28)	3.52 (8)
12.0°	-2.0 (14)	5.03 (25)	4.51 (27)	8.40 (17)	9.81 (6)
15.0°	-2.70 (11)	4.95 (16)	5.81 (16)	8.59 (12)	8.83 (9)

Table 4.V(b) First Iteration  $\Delta x=5.0, \Delta y=-4.0, \Delta \theta=11.0^\circ, \kappa=.84$ 

Scaling Factor (column estimates)					
Angle	.75	.80	.85	.90	.95
3.0°	2.18 (12)	-9.48 (9)	-7.71 (8)	-1.26 (9)	-3.73 (6)
6.0°	1.67 (11)	-4.25 (13)	-7.89 (11)	-8.76 (7)	-8.98 (5)
9.0°	-.79 (16)	-.906 (29)	-6.30 (22)	-5.90 (10)	-9.39 (7)
12.0°	-2.47 (14)	-3.21 (18)	-2.39 (15)	-2.93 (10)	4.33 (8)
15.0°	-6.15 (10)	-2.74 (12)	-3.06 (10)	-3.16 (7)	-2.04 (7)

Table 4. VI(a) Second Iteration  $\Delta x=5.0, \Delta y=-4.0, \Delta \theta=11.0^\circ, \kappa=.84$ 

Scaling Factor (row estimates)					
Angle	.79	.81	.82	.83	.84
8.5°	7.09 (6)	4.79 (23)	4.41 (43)	1.92 (54)	1.70 (76)
9.5°	-7.18 (4)	6.27 (30)	3.0 (64)	1.97 (106)	1.34 (93)
10.0°	3.97 (10)	5.78 (32)	2.90 (73)	1.83 (191)	1.33 (104)
10.5°	1.78 (11)	3.77 (28)	2.53 (67)	2.17 (89)	0.58 (73)
11.0°	4.08 (11)	5.17 (35)	-6.42 (22)	1.76 (61)	2.94 (22)

Table 4. VI(b) Second Iteration  $\Delta x=5.0, \Delta y=-4.0, \Delta \theta=11.0^\circ, \kappa=.84$ 

Scaling Factor (column estimates)					
Angle	.79	.81	.82	.83	.84
8.5°	-3.30 (16)	-1.38 (18)	-2.70 (26)	-2.21 (28)	-1.01 (32)
9.5°	-6.33 (25)	-3.78 (28)	-3.44 (36)	-2.36 (43)	-.103 (30)
10.0°	-6.40 (20)	-3.72 (39)	-3.09 (40)	-1.94 (62)	-2.19 (39)
10.5°	-6.44 (21)	-3.95 (34)	-3.55 (36)	-2.62 (43)	-2.15 (37)
11.0°	-6.96 (16)	-4.23 (27)	-7.30 (16)	-2.93 (34)	-6.22 (13)

Table 4.VII A summary of estimates in the rectangular plane of a real image

Rectangle Plane Estimates		
No. of Iteration	row ( $\Delta\hat{x}$ )	column ( $\Delta\hat{y}$ )
1	2.67	-0.906
2	1.83	-1.94
3	0.065	-0.78
4	-	-0.208
-	$\sum_{j=1}^4 \Delta\hat{x}_j = 4.565$	$\sum_{j=1}^4 \Delta\hat{y}_j = -3.834$

maximum frequency spectrum must be then located in order to compute the perturbations. The following is a number of operations required to compute the correlation.

1. Number of operations for FFT =  $2n \log_2 n$
2. Number of operations for complex matrix multiplication:

$$\text{Number of additions} = n^2(n-1)$$

$$\text{Number of multiplications} = n^3$$

#### 4.4.2 One-Dimensional Correlation

The basic algorithm for the one-dimensional algorithm is indicated in (12). If there is no row search performed, then for one line we have

$$\text{Number of multiplications} = 3n$$

$$\text{Number of additions} = 4(n-1) + 1$$

$$\text{Number of division} = 1$$

Thus, for  $n$  lines, we have

$$\text{Number of multiplications} = n(3n)$$

$$\text{Number of additions} = 4n(n-1) + n$$

$$\text{Number of divisions} = n$$

For 2D motion, the total number of operations are

$$\text{Number of multiplications} = 2 \times n(3n) = 6n^2$$

$$\text{Number of additions} = 2 \times (4n(n-1) + n) = 8n(n-1) + 2n$$

$$\text{Number of divisions} = n + n = 2n$$

*Example.* Let  $n$  equal to 256. For direct correlation, we have

$$\text{FFT computation} = 2n \log_2 n = 4096$$

Matrix multiplications:

$$\text{Number of additions} = n^2(n-1) = 16,711,680$$

$$\text{Number of multiplications} = n^3 = 16,777,216$$

For one-dimensional correlation, we have

$$\text{Number of multiplications} = 6n^2 = 393,216$$

$$\text{Number of additions} = 8n(n-1) + 2n = 522,752$$

$$\text{Number of divisions} = 2n = 512$$

The percentage of number of operations used is

$$\text{Multiplications} = \frac{393216}{16777216} \times 100 = 2.344\%$$

$$\text{Additions} = \frac{522752}{16711680} \times 100 = 3.128\%$$

As can be seen, we save over 95% with the one-dimensional correlation. This does not count computation time for FFT since the number of operations required for the FFT computation can be used to offset the number of divisions. Furthermore, complex matrix multiplication usually takes more computation time than a normal one.

The above analysis applies to binary images since all rows have the same intensity function. Furthermore, it also applies to the case of one-dimensional motion regardless of the image type. Thus, no row search is required for both cases. For a grey level image, row search is necessary since a different row has a different intensity function. Column search, however, is not necessary. The reason for this is that once the position where most consistent estimates in row direction occur, have been located, a column displacement can be obtained. This is true because a row mismatch causes by a vertical shift or column displacement. One may apply the one-dimensional correlation in the column direction in order to confirm the vertical displacement as we did to all of the simulations.

Let's include a row search in the analysis above. Let's say that we will search  $\pm 10\%$  of an object dimension. Notice that the second term in (12) only needs to be computed one time. Therefore, for  $n$  lines we have

$$\begin{aligned}\text{Number of multiplications} &= n \{ 2n(.2n) + 3n \} \\ \text{Number of additions} &= n \{ 3(n-1)(.2n) + 4(n-1) + (.2n+1) \} \\ \text{Number of divisions} &= n(2n+1)\end{aligned}$$

For  $n$  equals to 256, we have

$$\begin{aligned}\text{Number of multiplications} &= 6,907,494.40 \\ \text{Number of additions} &= 10,301,491.2 \\ \text{Number of divisions} &= 13,363.2\end{aligned}$$

The percentage of number of operations used is

$$\begin{aligned}\text{Multiplications} &= \frac{6907494.40}{16777216} \times 100 = 41.17\% \\ \text{Additions} &= \frac{10301491.2}{16711680} \times 100 = 61.64\%\end{aligned}$$

Again, the number of divisions are used to offset with the number of operations for the FFT and the additional time required for the complex number multiplication and addition. As can be seen, we do not save as much as when we compute the perturbations without a row search. However, we still save 40% or more of the computation time. The point that we would like to make here is that as the motion gets more complicated, the processing time increases tremendously. Thus, the parallel processors are necessary in order to achieve a real-time processing for 3D motion. A single processor will not be able to accomplish the task. Although it is true that the conventional correlation algorithm as well as other algorithms can be implemented in highly parallel fashion, most of them are too complex. Thus, it is very difficult to design in such manner. However, the one-dimensional correlation has the property of simplicity and separability. That is, its mathematic formula is simple. Each row of a moving area can be computed independently. Thus, it is easy to design and implement the one-dimensional correlation algorithm in highly parallel fashion (see [17] for hardware implementation of the algorithm).

#### 4.5. Conclusion

A row (column)-correlation-algorithm for motion prediction, easily implementable by parallel architecture, has been developed. If implemented by means of a general purpose digital computer, time consuming process is expected. However, it is still faster than conventional correlation technique. The algorithm also combines many good features of various algorithms presented in the literature. For example, it has correlation feature of [19] and spatio-temporal technique of [15,16]. In addition, the algorithm also possesses a good noise immunity property. The computation of gradients around edges or corners does not affect the accuracy of the estimates. The

convergence rate appears to be fairly quick for a large motion. The combination of the algorithm and the BVS sensor with two planes configurations has solved the 3D motion problem. The results from the real image simulation are encouraging.

## REFERENCES

- [1] Braccini, C., Gambardella, G., Sandini, G. & Tagliasco, V., "A Model of the Early Stages of the Human Visual System : Functional and Topological Transformations Performed in the Peripheral Visual System," *Biological Cybernetics*, Vol.44, 1982, pp.47-58.
- [2] Messner, R.A. & Szu, H.H., "Simultaneous image processing & feature extraction for 2D non-uniform sensors," *Proc. SPIE*, Vol.449, 7-10 Nov. 1983, pp.693-710.
- [3] Weiman, C. & Chaikin, G., "Logarithmic Spiral Grids for Image Processing and Display," *Computer Graphics and Image Processing*, Vol. 11, 1979, pp.197-226.
- [4] Minnix, J. I., "Structural Analysis and Design of a Biologically Based Vision Sensor for Machine Vision," Master Thesis, Univ. Virginia, Charlottesville, August 1986.
- [5] Schultze, M., "Zur Anatomie und Physiologie der Retina," *Arch. Mikrosko. Anat.*, Vol. 2., 1866, pp. 175-286.
- [6] Trigo, R.M. et al., "A Biological-Structure Visual Sensor for Robotics Applications," *Proc. of 16th ISIR*, Brussels, Bel. Sept. 1986.
- [7] Trigo, R.M. et al., "A New Sensor for Machine Vision," *Proc. of IFAC Int. Conf. on Low Cost Sensors*, Valeucia, Spain, Nov. 1986.
- [8] Kaas, J.H., "The Organization of Visual Cortex in Primates," *The Primates Nervous System* by Noback and Lucaett. Vol.8, 1978.
- [9] Kass, Michael, "Computing Visual Correspondence," *Image Understanding: Proceedings of a Workshop (14th) Held at Arlington, Virginia on June 23, 1983*, AD-A130 251.
- [10] Zse-Cherng Lin and others, "Motion Estimation From 3-D Point Sets With And Without Correspondences," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 194-201 (1986).
- [11] Webb, J. A. and Aggarwal, J. K. , "Shape and Correspondence," *Computer Vision, Graphics, and Image Processing* 21, 145-160 (1983).
- [12] Aloimonos, J. and Basu, A. , "Shape and 3-D Motion from Contour without point to point Correspondences:General principles," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 518-527 (1986).
- [13] Kanatani, K., "Tracing Planar Surface Motion from Projection without Knowing the Correspondence," *Comp. Vision, Graphics & Image Process.*, 29, pp.1-12, 1985.
- [14] Paquin, R. and Dubois, E., "A Spatio-Temporal Gradient Method for Estimating the Displacement Field in Time-Varying Imagery," *Computer Vision, Graphics, and Image Processing* 21, 205-221 (1983).
- [15] Netravali, A.N. and Robbins, J.D., "Motion-Compensated Television Coding: Part I," *Bell Syst. Tech. J. (USA)*, vol.58, No.3 pp.631-70, 1979.
- [16] Cafforio, C. and Rocca, F., "Methods for Measuring Small Displacements of Television," *IEEE Trans. Inform. Theory*, vol. IT-22, pp.573-579, Sept. 1976.
- [17] Narathong, C., "An Algorithm for Motion Prediction Using a Biological Visual Sensor (BVS)," Ph.D. Dissertation, Univ. of Virginia, Charlottesville, August 1987.

- [18] Orban, G.A., Neuronal Operations in the Visual Cortex. New York: Spring-Verlag, 1984, Ch.7.
- [19] D. R. Sullivan and J. S. Martin, "A Coarse Search Correlation Tracker for Image Registration," IEEE Transactions on Aerospace and Electronic Systems," Vol. AES-17. No.1, Jan 1981.

## 5. PATTERN RECOGNITION USING THE BVS, THE C-TRANSFORM AND A NEUTRAL NETWORK CLASSIFIER

### 5.1. Introduction

A major problem in pattern recognition is object recognition when position and orientation vary in the image field. Typically, the problem is solved by template matching. The template is compared to the image (usually by taking some sort of correlation), and it is then shifted to another location and again compared. This process is repeated until all possible locations are covered. If more than one object is sought, then more templates must be used. The case with the highest correlation is then selected as the recognized object. This process, while fairly accurate, is very time consuming even when small image arrays are used.

In the biological visual system (BVS) a complex logarithmic conformal mapping takes place between the retina and parts of the cortex (specifically area 17). The outstanding feature of the BVS sensor is form invariance under magnification and rotation in computation plane. The use of a translation invariant transform in computation plane would shift the object to a standard location regardless of the rotational orientation or size of the object in image plane. This would allow faster pattern recognition of objects which is not dependent on rotations and magnifications.

A class of translation invariant transforms, the C - transforms, possesses the desired characteristics plus other features which make this class useful. The functions involved are very simple, and the transforms themselves exhibit a high degree of parallelism. This transform can be performed using a highly parallel artificial neural system (ANS) for greater speed and reliability.

This class of transforms uses identical functional layers of simple operations that can be done very quickly without computationally intensive multiplications and divisions. Using this parallel structure, the transforms provide a method of transforming any shifted image to a standard form, that can be used with a template matching scheme for pattern recognition. The major advantages of such a system are its simplicity, highly parallel nature, fast operation, and the computational gain derived from the reduced complexity of the template matching.

### 5.2 Background

The C-transform class is a class of nonlinear translation invariant transforms that can be implemented with simple parallel networks of computational elements. Reitboeck and Brody (1969)[1] developed the R-transform, the first known member of this class. The R-transform could be implemented using simple functions (sum and absolute difference) in a parallel arrangement similar to that of the Fast Fourier Transform (Fig. 5.1).

The transform can be used on vectors of length  $2^n$  where  $n$  is an integer. The actual calculation of the transform can be expressed as:

Symmetric Functions  $f_1$  and  $f_2$  over sequence:

$$a(l) \quad , \quad l=0,1,\dots,2^n-1 \quad (1)$$

The  $K^{th}$  component of the transform:

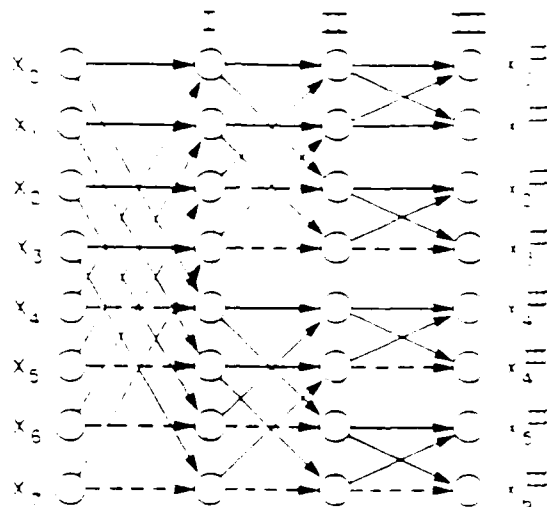


Figure 5.1. C-Transform Flow Graph [2]

$$A(K), K=0,1,\dots,2^n-1 \quad (2)$$

let the  $n$ -bit binary expansion of  $K$  be  $k_0 k_1 \dots k_{n-1}$ ;  
find sequences  $y_0(I), y_1(I), \dots, y_n(I)$  by:

$$y_0(I) = a(I) \quad (3)$$

and

$$y_{r+1}(I) = f_{k_r}(y_r(I), y_r(I + 2^{n-(r+1)})) \quad (4)$$

where

$$f_0 = f_1 \text{ if } k_r = 0 \quad (5)$$

or

$$f_1 = f_2 \text{ if } k_r = 1 \quad (6)$$

The  $K^{\text{th}}$  component of the transform is then:

$$A(K) = y_n(K) \quad (7)$$

This is the formulation of the transform for the one dimensional case. The extension into two dimensions provides a similarly simple representation of the transform:

Let  $A(I, J); I, J=0, 1, \dots, 2^N-1$  be an array.

Construct matrices  $Y^0, Y^1, \dots, Y^N$  such that:

$$Y^0(I, J) = A(I, J); I, J=0, 1, \dots, 2^N-1 \quad (8)$$

Let

$$Y^{r-1}(I, J) = P \quad (9)$$

$$Y^{r-1}(I+2^{N-1}, J) = Q \quad (10)$$

$$Y^{r-1}(I, J+2^{N-1}) = R \quad (11)$$

$$Y^{r-1}(I+2^{N-1}, J+2^{N-1}) = S \quad (12)$$

Then form matrices by:

$$Y^r(2I, 2J) = f_1(f_1(P, Q), f_1(R, S)) \quad (13)$$

$$Y^r(2I, 2J+1) = f_2(f_1(P, Q), f_1(R, S)) \quad (14)$$

$$Y^r(2I+1, 2J) = f_1(f_2(P, Q), f_2(R, S)) \quad (15)$$

$$Y^r(2I+1, 2J+1) = f_2(f_2(P, Q), f_2(R, S)) \quad (16)$$

for  $I, J=0, 1, \dots, 2^{N-1}-1$

The Two-D transform is then:

$$X(I, J) = Y^N(I, J); I, J=0, 1, \dots, 2^N-1 \quad (17)$$

Wagh and Kanetkar (1975)[3] examined the transform in more detail and developed a form called the Generalized R-transform. They also made an extension of the two dimensional R-transform into arrays that were rectangular, not only the square arrays of Reitboeck and Brody.

Wagh and Kanetkar (1977)[2] generalized this translation invariant transform from the R-transform's sum and absolute difference functions to any set of two functions that are argument symmetric. They make a convincing case for the use of the M-transform on the grounds that at higher order it has greater accuracy (more possible classes) than the R-transform. The M-transform also provides binary outputs for binary inputs, which leads to a smaller transform volume (and therefore less memory used) as compared to the R-transform which does not provide binary outputs. They also claim that the functions used in the M-transform (logical OR and AND) are faster to implement than the corresponding R-transform functions. The binary nature of the M-transform makes it

more attractive for ANS implementation.

Burkhardt and Müller (1980)[4] did a much more in depth mathematical examination of the properties of this class of transforms. They included the R and M transforms as well as two others in their analysis of the translation invariant mapping properties of the transform class.

Reitboeck and Altmann (1984)[5] discuss the use of the R-transform in a detailed model of magnification and rotation invariance in the BVS. They compare the R-transform to other methods (Mellin and Fourier-Mellin transforms) as they could possibly exist in the BVS. Their model covers the log-polar mapping of information, the DOG smoothing function that is similar to the receptive field weighting in the retina, and various processing (thresholding and edge detection) that is useful in obtaining reasonable results. They also suggest that these transforms can be implemented using an ANS. They go to a great length to justify the use of these transformations in pattern recognition in their detailed BVS model.

### 5.3. R and M Transform Advantages and Disadvantages

The R and M transforms exhibit a number of advantages and disadvantages that are relevant to pattern recognition problems. These will also influence choices that are made concerning implementation.

An advantage demonstrated by the R-transform, as mentioned earlier, is that it can work on continuous as well as binary information. This would allow grey level processing of images under certain circumstances. Another advantage is that the functions involved, sum and absolute difference are accomplished rather quickly in computer implementations.

The R-transform also has a number of disadvantages, one of the most notable being that the transform volume grows very quickly. This means that the numbers in the transformed vector or matrix are typically much larger than the values in the input vector or array. This makes it necessary to use more computer storage space to perform this transform. Another aspect of this problem is that this characteristic leads to non-binary outputs for binary inputs. This could lead to difficulties in pattern recognition schemes. In higher order problems, the R-transform has fewer distinct classes of transforms than the M-transform. This means that the accuracy of transformation is lower for the R-transform when the problem's order is high.

The M-transform has several advantages when compared to the R-transform. The transform volume of the M-transform is much more limited than the R-transform and is on the same order as the input volume. The functions involved, logical AND and OR, are usually among the fastest implemented on the computer, and are probably faster than the corresponding R-transform functions. The M-transform provides binary outputs for binary inputs, so there is no compatibility problems with other parts of a pattern recognition system. In low order problems, the M-transform has fewer distinct output classes than the R-transform, but as the input order increases this changes so that the M-transform is more accurate.

The major disadvantages of the M-transform include the necessity for binary input information. The M-transform cannot use continuous data. Another disadvantage is due to the functions, OR and AND, which are sometimes difficult to implement using some

higher level programming languages.

#### 5.4. Computer Simulation Results

Initial testing was done on the R and M-transforms by generating a large number of 8 dimensional binary vectors. This vector set was used to test the properties of the transforms. This was done primarily to determine if the theory was correct in all of its claims.

The R and M-transforms were both tested with these vectors and shifted vectors, including cyclic shifts, producing the same transformed pattern, as expected. Advantages and disadvantages of both transforms, mentioned earlier, were also confirmed. Further testing was performed on 8 dimensional binary vectors, this time on a set of all 256 possible 8 dimensional vectors. The major reasons for this experimentation was to determine transform volume and the number and types of classes present in the output of the R and M-transforms.

The R-transform was performed on this set of 256 vectors and the results in terms of translational invariance were unchanged. The maximum input vector volume was 8 (for the all 1s case) and the maximum output transform volume was 20. The output was not binary just as in the preliminary testing. The number of distinct classes found was 21, so the R-transform divides the 256 8 dimensional binary vectors into 21 classes.

The M-transforms of this vector set again demonstrated the invariance to translations as shown by the previous inquiry. The maximum input vector volume was 8 and the maximum output vector volume was also 8. The output was binary, and the transformed vector is identical to the input vector (shifted to a standard location). The transform yields 20 distinct classes of outputs for the 256 binary vector inputs.

In comparison, the time performance is very similar for the transforms, both of which are very fast. The transform volume is much greater for the R-transform, and this problem will continue to worsen as the order of the problem increases. The R-transform provides an output that is continuous while the M-transform's output is binary. The R-transform does not resemble, even superficially, the input, while the M-transform is identical (but often shifted) to the input. The number of distinct classes at this low order is slightly greater for the R-transform, but as the order increases this will reverse and the M-transform will become more accurate.

#### 5.5. Simulation of 2 Dimensional Case"

The next stage of simulation was done to test the 2 dimensional R and M-transforms. 8x8 binary arrays were used as inputs to the 2 dimensional transforms. The images tested were binary 4x4 squares in the 8x8 array. The square was shifted to 25 different locations in the 8x8 array to determine the translational invariance of the transforms.

The R and M-transforms gave the same output for all 25 4x4 squares, which verifies the invariance to translations of these transforms. The volume of the R-transform was quite large compared to the input volume. The M-transform volume was the same as the input volume, and much smaller than the R-transform volume.

The R and M-transform performance was examined for higher order 2 dimensional arrays. The purpose was to determine the characteristics of the transforms, specifically to

verify properties and to test the sensitivity of the process. Three classes of objects were considered, lines, squares, and circles. The images contained one of the three at some location in the image field. The sensitivity of the transforms was tested by varying the width of the lines and object sizes slightly and examining the differences in the resulting transforms. As before, the images were binary, and the array size was  $64 \times 64$ . This size was chosen because it meets the  $2^n$  requirement and is easily done using the BVS sensor.

The R-transform of the higher order arrays again demonstrated the shift invariant nature of the transform. The continuous output had a volume much greater than either the input volume or the lower order simulations. The output arrays had some sensitivity to line width and object size, i. e. slight variations gave slightly different transforms. This variation in the transform increases in proportion with the variation in the image. This sensitivity, while bothersome, is certainly within limits of adjustment for pattern recognition. The actual recognition algorithms must be adjusted to tolerate some variations.

Tests using the M-transform provided similar verification of the translational invariance properties. The output in this case was binary, and the volume was of the same order as the input volume. The M-transform, like the R-transform, showed some sensitivity to line thickness and slight variations of size, but as before, these could be compensated for in the recognition algorithms. The output, unlike the R-transform output, looks like the object in the input array.

There are several problems with using arrays of this size, primarily due to memory space usage. While binary images take up much less space than grey level images, the size of these ( $64 \times 64$  or larger) prevents comparison of more than a small number at once. This immediately eliminates certain types of pattern recognition schemes. Another problem is that even after the scene has been "standardized" by the transform, it still poses a pattern recognition problem of fairly high order, so even though the processing is drastically simplified it is still fairly extensive.

## 5.6. Use of Transforms With BVS Sensor

The R and M transforms must be used on square or rectangular arrays so the only compatible computational plane is the arc of ring arrangement. This gives a rectilinear computational plane that can be adjusted to the proper square  $2^n$  array size.

An advantage of the transforms with the BVS sensor, as mentioned earlier, is that under the LSM scalings and rotations become translations, so these transforms can be used to standardize all magnifications and rotations to a single image for template matching.

The choice of the C-transforms, specifically the M-transform, for ANS implementation is motivated by a number of reasons. The primary reasons, as mentioned above, are the simplicity of function and parallelism. The transforms can be performed by identical layers of simple elements. Another aspect which reinforces the suggestion of ANS implementation is that the transforms have a known input/output relationship, so performance can be evaluated easily. The M-transform uses only binary information for the vector or image representation, so the high gain sigmoid transfer function used in many ANS will provide a binary output. And finally, the simulations can be developed in fairly low order and can then be easily generalized to two dimensions or to higher

order problems. These traits of the M-transform make it a reasonable candidate for ANS implementation.

### 5.7. Network Structure and Weight Matrix

An ANS implementation of the M-transform utilizes the processing elements (PEs) as threshold logic gates to perform the functions of AND and OR. This means that a high gain is necessary to make them behave properly. The connection weights can be determined quite easily for the vector case, since each PE has two inputs.

In the AND case, the PE should go to one if and only if both inputs are one, so the inputs should be weighted such that their weighted sum is greater than 0.5, but their individual weights are less than 0.5. A selection of 0.4 for the AND PE case should provide this function. In the OR PE case, the output should be one if any input is one, so the weights should be chosen such that any input of one will give an element input of more than 0.5. The choice was 0.6 for the OR PE weightings, which made it possible for the weighted sum of the inputs to be greater than one. This made choice of the transfer function harder.

The form of the network can be seen as Fig. 5.2.

As shown, the network is not symmetric, as is the case in a Hopfield network. But in this case there is no feedback, so stability is assured. Finite inputs yield finite outputs. The binary nature of the solution can be demonstrated as:

$$\text{Inputs : } i_j \leq 1 \text{ so} \quad (18)$$

$$\text{Element inputs : } x_i = \sum_j w_{ij} \leq 1.2 \quad (19)$$

$$\text{Element outputs : } V_i = g(x_i) \leq 1 \quad (20)$$

$$\text{Element inputs : } X_i = \sum_j v_{ij} \leq 1.2 \quad (21)$$

$$\text{Final outputs : } m_i = g(X_i) \leq 1 \quad (22)$$

In the demonstration above,  $\leq 1$  means that the element outputs that should be one are very close to one due to their input values and the sigmoid transfer function. The transfer function must have the property of mapping values of input that are greater than 1 to an output of 1 (or very nearly 1), as well as the normal sigmoid characteristic about 0.5. This is accomplished by limiting the input side of the PEs to only small values greater than 1 (in this case 1.2) and to use only even powers of gain. The transfer function actually used is:

$$V(1,j) = g(x(1,j)) = x(1,j)^p (1 - x(1,j)^p) + x(1,j)^p \quad (23)$$

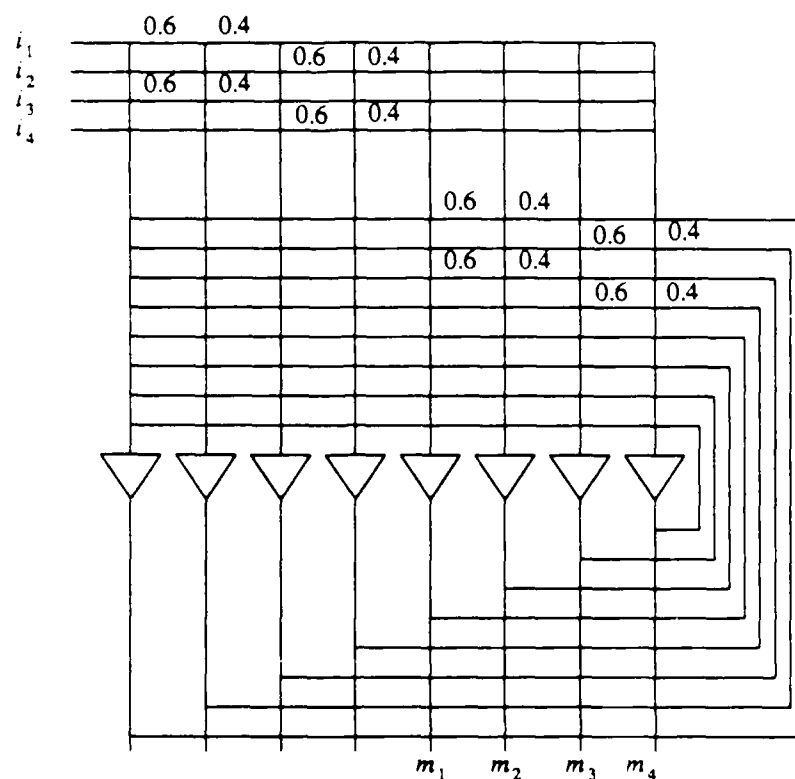


Figure 5.2. Network Structure

Where  $V(1,j)$  is the output of element  $j$  on slab 1,  $x(1,j)$  is the input to element  $j$  on slab 1 and  $p$  is the gain of the function. The evenness requirement is to prevent the output from becoming greater than 1. In order to make the network behave properly a number of other functions must also be specified. The I-functions relate the scaling of the incoming signals by the proper weights, and the F-function determines how the I-functions are combined. There are two I-functions, one corresponding to the external inputs to the elements (slab 0 to slab 1 connections), and the other corresponding to feedback to the elements (slab 1 to slab 1 connections). The functions used are shown below:

$$I(1,0) = V(0,i)w(1,j,0,i) \quad (24)$$

$$I(1,1) = V(1,i)w(1,j,1,i) \quad (25)$$

and

$$F(1) = I(1,0) + I(1,1) \quad (26)$$

With the network structure and weight matrix described above, the M-transform was performed on 4 dimensional binary vectors. We would expect to see the translational invariance of the transform, as well as very fast operation of the parallel arrangement. The epsilon step size can be 1 since the output is essentially a function of the input directly and not of the past as in the TSP. For all 4 dimensional inputs the results are summarized in the following table (Fig 5.3.):

This table clearly shows the translational invariance of the M-transform. In the case of a translated vector (for example 0001 and 0100) the output is the same as the non-translated case (both are 0001). This just as we expected from the previous discussion. The transform is also very fast, depending upon the choice of epsilon. The limit (epsilon of 1) is convergence in two steps due to the two layer parallelism of the transform.

The transform can also be expressed in a compressed form and extended into two dimensions.

### 5.8. Reduced Transform Representation

It is desired to develop a reduced representation for this transform so that it may be determined whether any computational savings can be obtained through another expression of the transform. This would involve using the PEs as more than just two input

M-transform Results : 4 Inputs	
Input	Output
0000	0000
0001	0001
0010	0001
0011	0011
0100	0001
0101	0101
0110	0011
0111	0111
1000	0001
1001	0011
1010	0101
1011	0111
1100	0011
1101	0111
1110	0111
1111	1111

Figure 5.3. 4 Dimensional Vector M-transform Results.

threshold logic gates. It seems possible that a compressed M-transform representation may use fewer elements or fewer layers than the normal arrangement.

The nature of the transform is examined and the equations for the outputs are determined in terms of the inputs. In the 4 input case, this procedure can be summarized as:

$$\text{Inputs} : I_1, I_2, I_3, I_4 \quad (27)$$

$$\text{Outputs} : M_1, M_2, M_3, M_4 \quad (28)$$

$$M_1 = I_1 + I_3 + I_2 + I_4 \quad (29)$$

$$M_2 = (I_1 + I_3)(I_2 + I_4) \quad (30)$$

$$M_3 = I_1 I_3 + I_2 I_4 \quad (32)$$

$$M_4 = I_1 I_3 I_2 I_4 \quad (33)$$

Where the "add" and "multiply" operations are logical OR and AND.

The above equations demonstrate that two of the functions,  $M_1$  and  $M_4$ , can indeed be compressed because they consist of only one type of function. The other two,  $M_2$  and  $M_3$ , cannot be done using a single element. Each of these is a comparison of two binary quantities that are in turn comparisons of two binary quantities. This means that since an element is needed for each comparison, 3 elements are necessary for each of  $M_2$  and  $M_3$ .

This total of 8 elements is the same as in the noncompressed case, so no computational advantage can be gained in the number of elements used. Furthermore,  $M_2$  and  $M_3$  are two layers deep, so there will be no time improvement either. Actually, the original case is superior because all of the elements reach the proper outputs at the same time, since they all have the same depth (number of levels).

This means that while expectations were for a more efficient use of network structure, the best arrangement proved to be the original parallel layers of identical structure.

### 5.9. Extension into Two Dimensions

The transform extension into 2 spatial dimensions can be accomplished in the manner described by equations 8-17. The transform flow in the 2 dimensional case consists of layers of arrays of elements, with each layer having the identical functional arrangement.

Unfortunately, the functions present in the 2 dimensional case are actually combinations of the simple functions of the 1 dimensional case (see equations 13-16). This leads to problems like those faced when a reduced transform representation was sought, i.e. it takes 3 PEs to make up one functional "element". This problem is not actually as bad as it first seems because the initial functions have quite a bit of overlap. This means that the output of the first PEs in a functional element is needed by 2 functional elements. This allows implementation of the 2 dimensional M-transform using only twice as many PEs as functional elements.

Given this requirement, the 4x4 2 dimensional M-transform can be performed using 64 PEs as 2 input threshold logic gates. The binary input signal enters from slab 0 and proceeds through two functional layers of 32 PEs each, which is actually four logical layers of 16 PEs each. This doubles the time to convergence for the M-transform.

The same functions, the g, I, and F-functions, are used in the 2 dimensional case as were used in the vector case. The weight matrix has been expanded and changed to reflect the 4 logical layers of the 2 dimensional transform. As in the previous implementation, the matrix is not symmetric, except in the sense that the two layers have symmetric functionality. Again there is no direct feedback, so stability is assured.

We would expect to see a transform that takes twice as long as the previous case, and provides the necessary invariance to translations. The results obtained do indeed demonstrate these two features, and the translational invariance is demonstrated in Fig. 5.4.

A number of other input arrays were used, and the results always showed the translational invariance property.

### 5.10. Implementation of Learning in an ANS

Neural networks in living creatures are often capable of learning quite a variety of things, which gives these networks a major advantage over conventional networks in a number of applications. Learning in a neural network is accomplished by the modification of the interconnection weights between the elements. An ANS that has the ability to learn must also include some provision for the modification of its connection weights. In a hardware device this may be difficult if not impossible from a practical standpoint, but it is easily accomplished in simulation.

The general form of the weight modification rule employed in ANS learning is:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

Where  $w_{ij}$  is the connection weight from unit  $i$  to unit  $j$ . This modification can be accomplished by three mechanisms used in determining  $\Delta w$ . They are broken into three groups according to the amount of supervision used in each.

The first type of learning rule is unsupervised, in which the network learns correlations, i.e. if element  $i$  on is likely to drive element  $j$  on the connection gets stronger. This rule can be summarized by the following equation:

$$\Delta w_{ij} = k_1 f(X, W_j) g(x_i, w_{ij})$$

In this rule,  $k_1$  is the learning parameter, which determines how much the connection weight is changed. This may be a constant or may vary over time.  $\Delta w$  is also a function of the current state of the  $j^{th}$  and the particular input and weight of the  $i^{th}$  element. Unsupervised learning is useful when the exact desired output is unknown, or there are many elements that cannot be directly specified.

The second type of learning is called weakly supervised and uses a single error signal to tell the network to learn or not to learn. This is like telling the network if it is wrong, but not telling it where it is wrong. This rule follows an equation like:

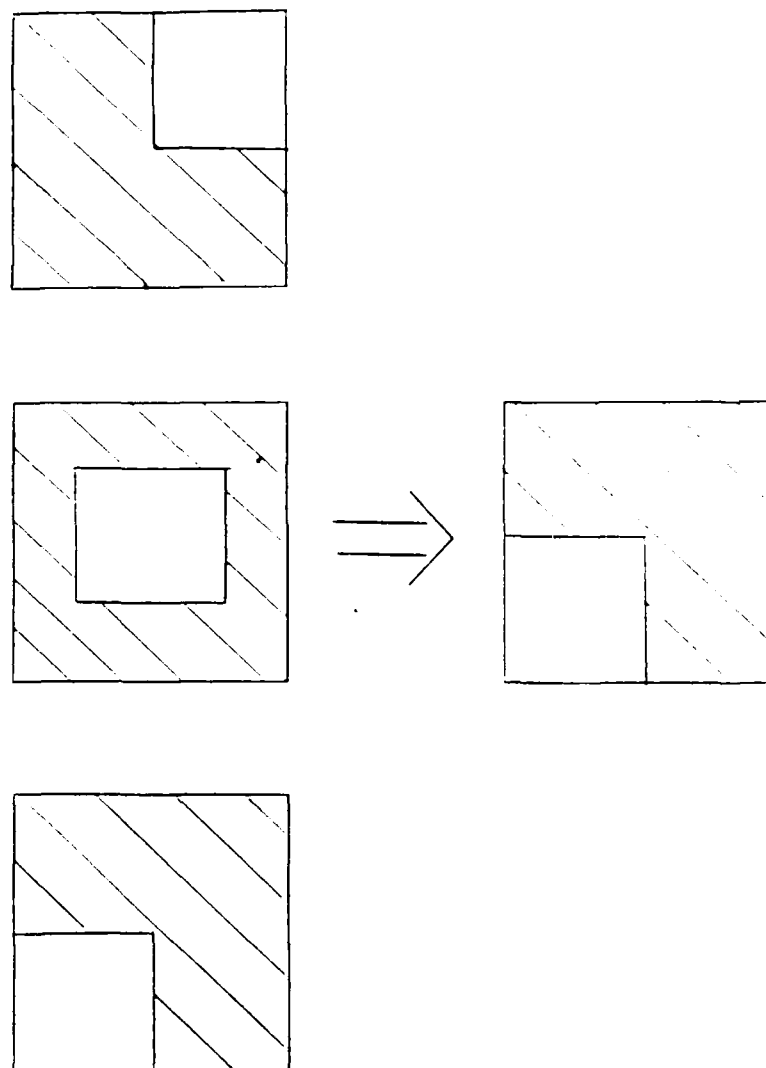


Figure 5.4. Two Dimensional M-Transform Results.

$$\Delta w_{ij} = k_1 f(I_i, w_i) g(x_i, w_{ij})$$

Where  $k_i$  is the learning parameter, as before. However, now  $\Delta w$  is a function of the error signal  $I_i$  and its weight  $w_i$ , as well as a function of the particular input and its weight. This type of learning is useful when the desired output is known, but for some reason the elements cannot all be directly specified (hidden elements) or a full error vector is unknown (order of the system is too large, etc.).

The third type of learning is highly supervised, in which the weight modifications are adjusted according to the amount of error, i.e. the weights are changed in proportion to their contribution to the error for that element. The general form of this rule is:

$$\Delta w_{ij} = k_i f(I_T, V_j) g(x_i, w_{ij})$$

Once again,  $k_i$  is the learning coefficient, but now  $\Delta w$  is a function of a training vector  $I_T$  and the output vector  $V$ , so that an error value is generated for each element. This differs from the previous weakly supervised case in which an overall error signal was generated. As before,  $\Delta w$  is some function of the individual weight and input for that element. This type of learning is used when the outputs of each element are known so that the error vector can be generated.

Since the M-transform can be calculated easily by a separate program, all of the outputs can be determined for the generation of the error vector. This allows the use of highly supervised learning to teach a simple network the M-transform. The  $\Delta w$  function that was used is:

$$\Delta w_{ij} = k_i (t_j - V_j) i_i$$

Where  $t_j$  and  $V_j$  are the desired and actual outputs of element  $j$  and  $i_i$  is the  $i^{\text{th}}$  input to element  $j$ . The difference corresponds to the  $j^{\text{th}}$  component of the error vector, and the  $i_i$  term determines the amount of contribution to the output for that particular weight. Using this function the network can be taught the M-transform.

The network structure used is the same as in the four dimensional vector case, as shown previously, with four inputs to eight elements. The I and F-functions are the same as before, and the g-function has been changed to the hyperbolic tangent function used by Hopfield. This function has been shifted and scaled appropriately so that 0 and 1 are the bounds as in our previous case. The new g-function was chosen because the old power function does not asymptotically approach 1 and 0. This means that if the input value becomes greater than 1, the output will become less than 1, so that the weight update rule will push the weight in the wrong direction. The hyperbolic tangent function doesn't have this problem.

The connection weights were initially all zero, as were the initial conditions. The network is presented with an input vector and allowed to reach a final state (two cycles). This is necessary so that the error vector is generated at the proper time. Once the final state is reached, the error vector is calculated from the training input and the element outputs. The weights are then updated according to the error vector, learning coefficient, and individual input. Then the initial condition is reset and the next input vector is presented and the process is repeated.

Since there are sixteen possible four dimensional input vectors, the cycle must be repeated a number of times to reach a working network. There are also a number of stability issues associated with the learning that influenced the final procedure. The gain of the sigmoid plays a significant part in the learning process. If the gain is too high, i.e. the sigmoid is too steep, all of the error components are either 0 or 1, and the system just goes from state to state never reaching the desired result. This forces a flatter sigmoid. Another factor in the stability is forcing the diagonal elements to remain zero throughout the simulation. This helps to maintain stability by removing direct feedback.

In addition to stability issues, there are several things that must be addressed to make the system behave properly. The first of these is the noise margin of the system, i.e. what values constitute zero and one. In this case, a noise margin of 0.15 was chosen so that values of .85 and above are 1 and .15 and below are 0. This seems to be a reasonable choice, and if the output was within this margin no learning takes place. Another important issue is the value of the learning parameter. A number of values, constant and variable, can be used, and several were examined to determine their effects on the learning algorithm.

The simulation was performed on the training set which consisted of the sixteen input vectors, each appended with an additional eight values representing the desired outputs of the elements. This was done for both constant and variable learning parameters. The constants used were 1.0, 0.5, 0.25, and 0.1. We would expect the lowering of the parameter to provide smoother paths in learning space, and for the system to take more iterations of the test set. The results obtained do indeed demonstrate this.

For the case of a unity learning parameter, the network learned the M-transform in about 12 iterations through the test set. The changes in the weights are very discontinuous or jumpy. When the parameter was lowered to 0.5, it required 30 iterations of the test set, but the weight changes were smoother. When the coefficient was lowered to 0.25, the number of iterations to the desired result jumped to nearly 100. The paths of the weight modifications were much more continuous. Finally, with a parameter of 0.1 the system took almost 1000 times through the test set to reach proper M-transform form, while the paths became very continuous in appearance.

The second case tested was for a varying learning coefficient. In this case, the parameter starts at 0.5 and is reduced in 0.1 increments every 10 cycles through the test set, stopping at 0.1. Results showed that this variable parameter provided a working network in around 60 cycles through the test set.

The weight matrix obtained by the learning algorithm is drastically different from that obtained from an initial design. This seems reasonable considering the nature of the learning process. An example of a final state of the weight matrix is shown in Fig. 5.5. In general, highly supervised learning seems to be an efficient and fairly quick method of teaching a neural-like network simple tasks such as the M-transform.

In general, the ANS approach to the M-transform is a simple, fast, and efficient method of performing the this translation invariant transform.

Once the images have been "normalized" to a standard location by means of a C-transform, the recognition problem remains to be solved. The complete system for pattern recognition is shown, in block diagram form, in Fig. 5.6.

---

Input Weight Matrix

1.099	0.799	1.099	0.799	0.000	0.000	0.000	0.000
3.199	-0.699	2.099	4.56e-05	0.000	0.000	0.000	0.000
0.599	1.099	0.799	1.099	0.000	0.000	0.000	0.000
-0.299	1.499	1.81e-05	1.399	0.000	0.000	0.000	0.000

Network Weight Matrix

0.000	8.16e-05	0.200	8.16e-05	0.200	8.16e-05	8.16e-05	8.16e-05
-3.199	0.000	-0.600	-0.699	-3.099	1.499	1.799	-1.499
7.71e-05	7.71e-05	0.000	7.71e-05	7.71e-05	7.71e-05	7.71e-05	7.71e-05
-0.499	-0.399	-0.999	0.000	-0.999	0.999	-1.099	-0.399
0.200	0.100	1.08e-04	1.08e-04	0.000	0.200	1.08e-04	1.08e-04
2.699	9.99e-02	3.099	-0.799	-1.299	0.000	-2.099	-0.899
-56.18	92.39	-58.08	95.38	35.30	-147.3	0.000	-3.699
-6.299	8.099	-7.099	8.598	-11.29	-2.999	4.899	0.000

---

Figure 5.5. Learned Weight Matrix.

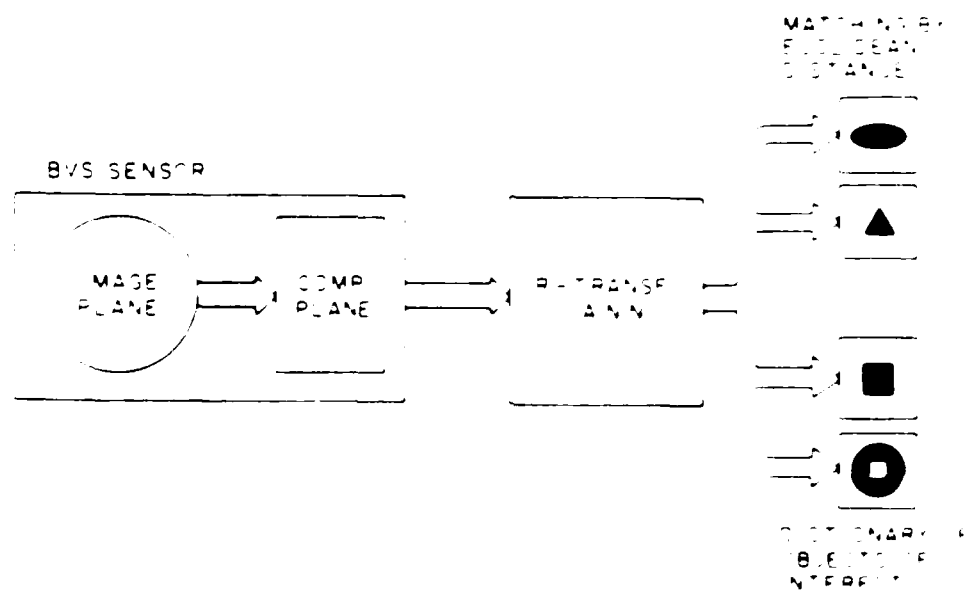


Figure 5.6 General Structure of Recognition System

Figure 5.6 General Structure of Recognition System

A Hopfield-like H&I-type network is the used for object recognition. In the input image, the network will identify the object of interest in the image. The output of the input is converted to a binary code, which is then used to identify the object of interest. The power of the network is to identify the object of interest in the image.

Hopfield and Tank are not adaptive, in other words, they lack a learning capability. The general form of the H&T network for pattern recognition is shown in Fig. 5.7[6].

The network has  $N$  inputs ( $x_1, \dots, x_N$ ),  $M$  outputs ( $y_1, y_2, \dots, y_M$ ), and an offset  $-V$ . Given a signal space  $\bar{x}$  which is spanned by a set of basis functions  $\bar{B}_k, k = 1, 2, \dots, M$ , find the best digital combination of basis functions  $\bar{B}_k$  which describe a given signal. For the specific problem of pattern recognition, the signal space  $\bar{x}$  would consist of the two dimensional array of pixels which make up the computation space image. For an  $8 \times 8$  array,  $N=64$ . The basis functions, in this case, would consist of 64-dimensional vector representations of the stored images. In the case of binary images, this would consist of ones in the positions where a pixel is excited in the stored image and zeros elsewhere. Each of the processing elements represents a basis function or, in our case, a stored image. If, after reaching a steady state, a processing element has value 1, then this element is the best match for the input signal. If the input image does not match any of the stored images, all the processing elements will have value zero in the steady state.

In designing the actual network, it is necessary to determine the value of the connection strengths. To this end, an energy function must be constructed which will have a minimum value for the "best" combination of basis functions describing a given signal. Hopfield and Tank have proposed the following general type of energy function [6].

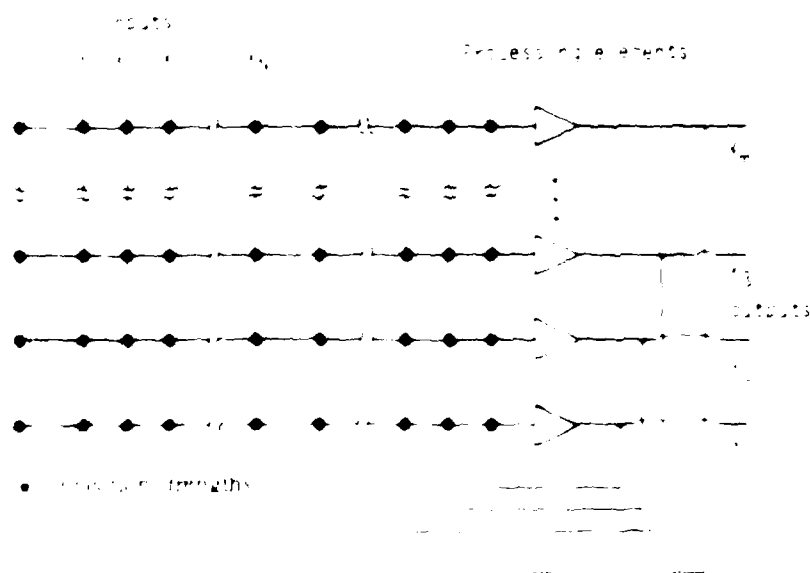


Figure 5.7 H&T Network for Pattern Recognition

$$E = 1/2(\bar{x} - \sum_k V_k \bar{B}_k)^2 + 1/2 \sum_k (\bar{B}_k \bar{B}_k)(V_k)(1 - V_k)$$

The first term is minimum when a combination of basis functions matches the input signal; the second term is minimum when the processing elements have values of zero or one. In terms of amplifier outputs, connection strengths and external inputs,

$$e = -1/2 \sum_{i=1}^N \sum_{j=1}^N T_{ij} V_i V_j - \sum_{i=1}^N V_i I_i$$

Comparing the two expressions,

$$T_{ij} = -(\bar{B}_i \bar{B}_j)$$

and

$$I_k = [\bar{x} \bar{B}_k - 1/2(\bar{B}_k \bar{B}_k)]$$

Note that this network can be used to recognize binary images which are non-translated, non-rotated, and non-scaled. In addition, the network is not self-adaptive. It can be used, in conjunction with the R or M-transform to implement a pattern recognition subsystem to recognize binary images in the computation plane of a HVS sensor which are invariant to scaling and rotation with respect to the optical axis.

### References

- [1] Reitboeck, H.J., and Brody, T.P., "A Transformation with Invariance under Cyclic Permutation for Applications in Pattern Recognition," *Inf. Control*, 15, pp. 130-154, 1969.
- [2] Wagh, M.D., and Kanetkar, S.V., "A Class of Translation Invariant Transforms," *IEEE Trans. on ASSP*, 25, pp. 203-205, 1977.
- [3] Wagh, M.D., and Kanetkar, S.V., "A Multiplexing Theorem and Generalisation of R-Transform," *Intern. J. Computer Math.*, (Sect. A) 5, pp. 163-171, 1975.
- [4] Burkhardt, H., and Müller, X., "On Invariant Sets of a Certain Class of Fast Translation-Invariant Transforms," *IEEE Trans. on ASSP*, 28, pp. 517-523, 1980.
- [5] Reitboeck, H.J., and Altmann, J., "A Model for Size- and Rotation-Invariant Pattern Processing in the Visual System," *Biol. Cybern.*, 51, pp. 113-121, 1984.
- [6] Tank, D.W., and Hopfield, J.J., "Simple 'Neural' Optimization Networks: An A/D Converter Signal Decision Circuit and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, CA3-33, No. 5, May 1986.

## 6. DUAL SENSOR IMPLEMENTATION AND ANALYSIS

### 6.1 Introduction

One of the major goals for machine vision systems is the development of an algorithm which can perform invariant pattern recognition in real time. One method which might be used to facilitate the solution of this problem is creating a vision system which obtains both a rectangular and a polar representation of images. The inherent characteristics of each of these geometries could be used to attain algorithms which can more easily achieve certain pattern recognition tasks.

A rectangular representation consists of a rectangular array of uniform sized square pixels. A representation of this type is desired for two main reasons. The first is that translations in this representation consist of simple shifts of the coordinate axes and, hence, translation invariant algorithms could be performed more easily. The second reason is that most existing pattern recognition and motion detection algorithms are based on a rectangular representation of images.

A polar representation consists of an array of pixels whose boundaries are determined by exponentially spaced concentric circles (for log-spiral pixels) and equally spaced rays emanating from the center. A representation of this type is desired because scalings and rotations result in simple shifts of the coordinate axes and, hence, scale and rotation invariant algorithms could be performed more easily with this representation. Another desirable attribute of this representation is that fewer pixels are required to describe a given image (since pixels in the periphery are larger). So, algorithms based on this representation should be faster.

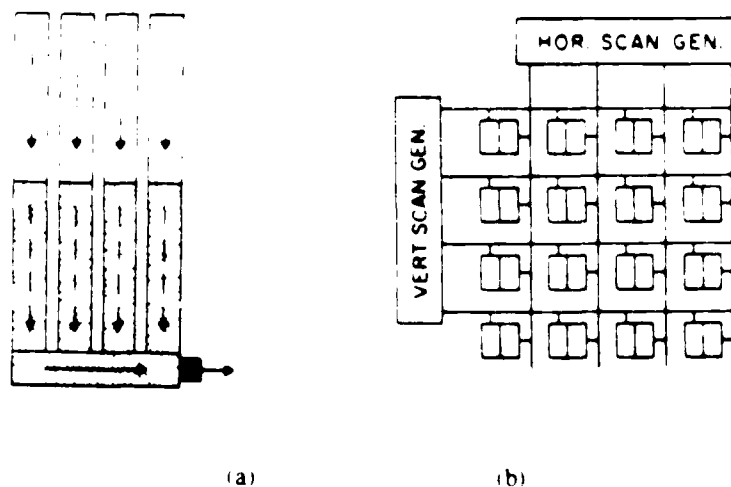
Given the unique advantages of each of these representations it is likely that an algorithm which utilized both of these representations could better perform invariant pattern recognition tasks. Hence, an important consideration is the means by which this dual representation of images can be obtained. This report discusses several possible methods which might be used, and includes an analysis of the error introduced by some of these methods.

### 6.2 Sensor Technology

In order to understand how a dual representation of an image can be obtained it is necessary to know something about the existing technology for obtaining digital representations of images. Area image sensors consist of a two dimensional array of photodiodes. Each photodiode represents a pixel in the image representation. In most cases these photodiodes are realized by charge-coupled devices (CCD's). There are two basic ways in which pixel values can be read out from an area image sensor. One way is line by line. A type of camera which uses this method is the frame transfer CCD imager. A second way is pixel by pixel. A charge injection device (CID) imager is capable of this method. Figure 6.1 shows the basic geometry for each of these camera types.

The individual columns of the frame transfer imager are separated by insulation. This forms the vertical boundary of the pixels. The horizontal boundaries of the pixels are maintained by voltage rails (not shown). The shape of these pixels is usually square or slightly rectangular. The image is integrated for one half a frame period (during which the photodiodes accumulate charge proportional to the amount of light which falls within

each photodiode's boundaries) and then the values are shifted rapidly (by the charge transfer capability of CCD's) into a storage area which is shielded from incident light. In this manner the smearing of images during read out is limited. Next, one pixel from each column is shifted to the output register which then reads out the values. This process is repeated so that pixels are read out line by line until all pixels are read out. In order to speed up this output process an output register could be provided for each column. In this way the values would still be read out line by line, however, it would be faster since each line would be read out at the same time.



(a) frame transfer CCD imager  
(b) CID imager

The pixels of the CID imager are insulated from each other on all sides. The shape of these pixels is also usually square or slightly rectangular. The pixel values can be read out one at a time in any order. A pixel is read out only when both its row and column are selected. This is achieved by voltage rails as seen in Fig. 6.1(b). An obvious drawback to this type of imager is that each pixel integrates the image at a different time. This effect can be minimized, however, by ensuring that the total read out time is small compared to the image integration time.

All standard CCD cameras produce image representations which are rectangular. However, this does not mean that other cameras could not be built which yield other representations directly. A novel area image sensor could be constructed to yield different representations simply by changing the boundaries which define the pixels. However, constructing a CCD camera with non-rectangular boundaries would probably pose some difficulties for chip fabricators, difficulties which may or may not be solvable.

For example, suppose a camera were desired which would yield a polar representation of images directly. The pixel boundaries would have to look as shown in Fig. 6.2. One problem which exists for this geometry is that pixel values would all have to be normalized since not all pixels are the same size. Second, a frame transfer imager probably could not be built with this geometry because there is no convenient place to transfer the

pixel values before reading them out. However, a CID imager could probably be constructed without much difficulty. The pixels could be addressed by constant radius and constant angle voltage rails.

It also might be possible that other novel image sensors could be constructed with many varying image representations being possible. Some of the solutions to obtaining a dual representation of images rely on this possibility.

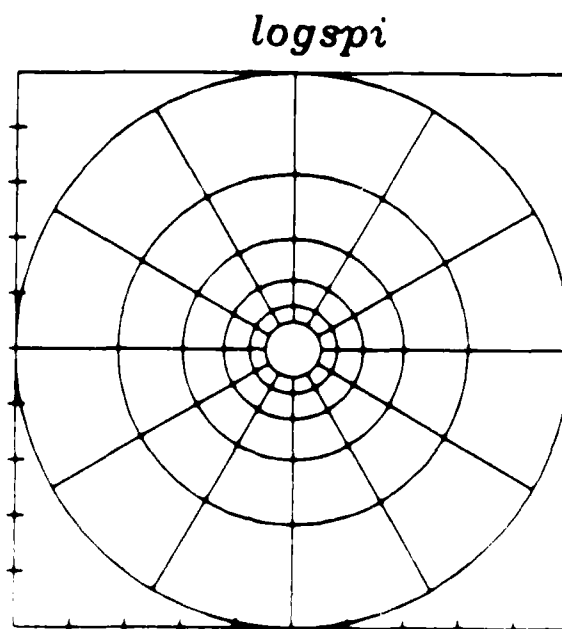


Fig. 6.2 Pixel boundaries for polar representation

### 6.3 Method for Obtaining Dual Representations of Images

Perhaps the best way of obtaining both a rectangular and a polar representation of images would be to devote an individual camera to each representation. A standard frame transfer CCD camera could be used to obtain the rectangular representation directly, and a novel sensor with its photosensors laid out in a polar format could be used to obtain the polar representation directly. This method would probably work, however, another method which would likely be less expensive and less bulky would be to use a single camera to obtain both representations. For example, a standard CCD camera could be used. This camera would yield the rectangular representations directly, and the polar representations could be obtained by combining rectangular pixels to form each individual polar pixel. The value of each polar pixel would be taken to be the average value of the rectangular pixels which have at least half their area within the polar pixel's boundaries. In this manner, both representations could be obtained from a single camera.

The camera used need not have its photosensors laid out in a rectangular format. It might be that a camera which has its photosensors laid out in a different format might be

able to better obtain the dual representations. For example, a camera with triangular pixels might be used. Triangular pixels could be combined to perfectly form rectangular pixels and could probably combine to form polar pixels better than rectangular pixels could. Thus, it might be advantageous to use a camera with triangular shaped pixels rather than rectangular shaped pixels.

It is clear from the above discussion that a disadvantage exists in using only one camera to obtain both a rectangular and a polar representation of images. Regardless of the shape of the pixels of the single camera, there will always be some error in trying to form both rectangular and polar pixels. The camera which minimizes this error might be deemed the best solution. In order to determine the camera type (determined by the layout of its pixels) which is best, a means of analyzing this error is needed. This error is examined in the next section.

#### 6.4 A Method to Determine which Solution is Best

The possible solutions being considered all require the superimposition of a computation plane (determined by the geometry of the representation desired) atop an image plane (determined by the geometry of the photosensors of the camera used), and the subsequent combining of image plane pixels to form the computation plane pixels. This combining is not exact, since the image plane pixels when combined together may not exactly form the computation plane pixel. Hence, this process can be said to add "noise" to the system. This "noise" can be quantified by considering two specific errors which arise from the process of combining image plane pixels to form computation plane pixels. These errors are as follows:

error one : Area included in computation plane pixel approximation which actually lies outside the pixel boundaries.

error two : Area not included in the computation plane pixel approximation which actually lies inside the pixel boundaries.

Figures 6.3 and 6.4 on the next page illustrate these two error types for the case when the image plane is rectangular and a log-spiral plane is superimposed on it.

An error value which will be proportional to the actual error of approximating a computation plane pixel with a combination of image plane pixels is

$$E = (\text{error one} + \text{error two}) / (\text{area of computation plane pixel})$$

The sum of error one and error two needs to be normalized by the area of the computation plane pixel being approximated since it is this ratio to which the possible error (in the computation plane pixel value) will depend on. Another fact to note is that the mapping technique of only including image plane pixels with at least one half their areas contained within the computation plane pixel being approximated tends to keep E minimal.

By examining the value of E for various image plane geometries, a means of determining a best solution is possible. The solution which minimizes E for the complete mapping (i.e. the sum of E for each computation plane pixel approximation) would be the

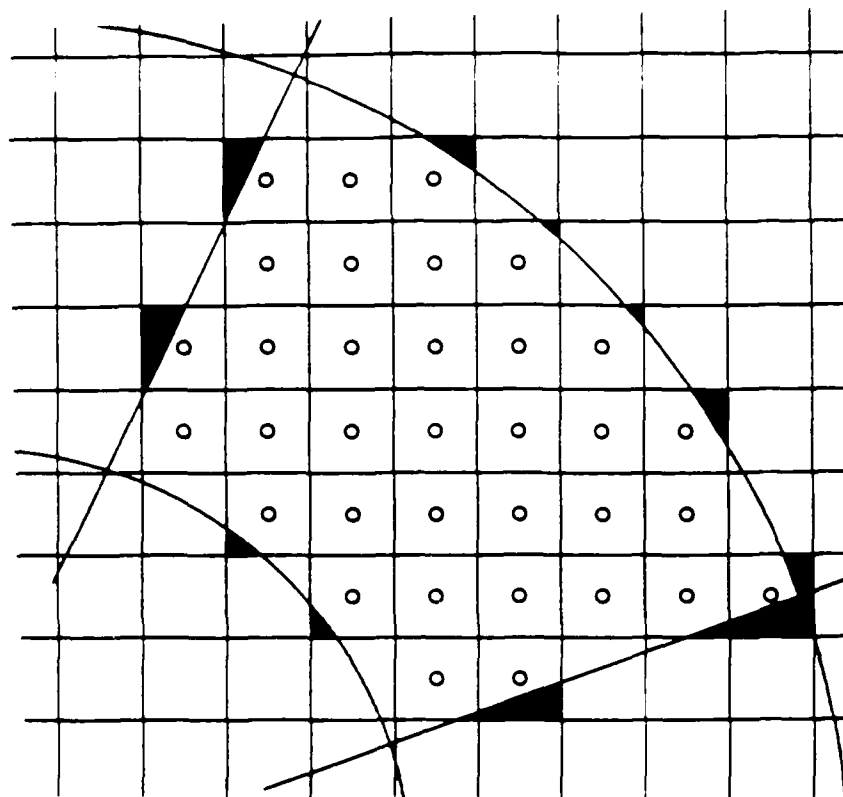


Fig. 6.3. Error (1): Area included in log spiral pixel approximation which actually lies outside pixel boundaries

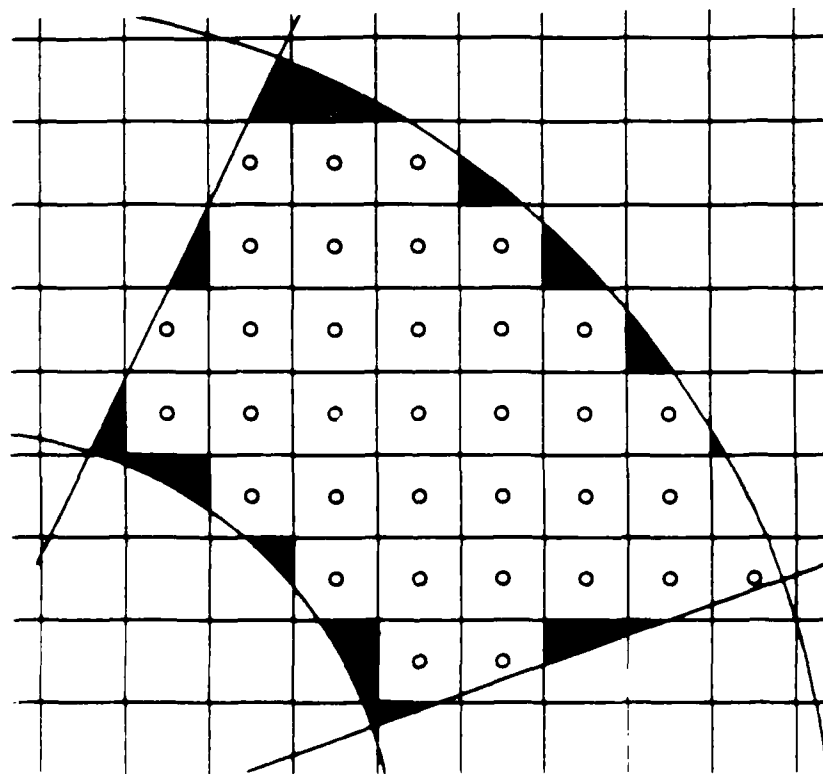


Fig. 6.4. Error (2): Area included in log spiral pixel approximation which actually lies inside the pixel boundaries

best solution.

The next section includes an analysis of the error variable  $E$  for different image plane geometries and computation plane geometries. The value of  $E$  is determined by computing areas for error one and error two for each computation plane pixel approximation.

### 6.5 Analysis of Error for Possible Solutions

The first solution to be examined is the case where the camera used is a standard CCD camera with square pixels. This solution is desirable for two main reasons. The first is that many cameras are available which have square pixels, thus a new camera would not have to be developed. The second reason is that square pixels can be combined to form polar pixels with little extra circuitry needed. Thus, if it is found that the error introduced by approximating polar pixels with combinations of square pixels is not so large that the polar representation is unacceptable, then this solution would probably be recommended, due to its ease of implementation.

In order to determine the amount of error introduced by this method a Fortran program was written to calculate this error. The program first determines the square pixels which have at least one half of their area within the given polar pixel and then determines the value of error one and error two (defined in previous section) for each polar pixel approximation. In calculating these values a slight approximation is used. This approximation is illustrated by the diagram in Fig. 6.5. This approximation will have only a very small effect on values of error calculated.

The results of several computations are included in graphs one through five in Appendix 6.A. A summary of these results is given below.

The geometry of the image plane and the computation plane for graphs one, two, and three was:

image plane : 512x512 array of square pixels ( $-5 < x, y < 5$ )  
 computation plane : log-spiral array, 36 rings and 36 pixels  
                           per ring. Radius of inner ring=0.5, and  
                           radius of outer ring=5.0.

#### Graph One :

This graph shows how error one, error two, and error one + error two vary with radius. For each ring the average value of these quantities was determined and plotted. From this graph it can be seen that error one and error two tend to stay very close in value. This is expected since pixels are included only if they have one half their area within the computation plane pixel. Also, it can be seen that the values all tend to increase with increasing radius. This is due to pixel sizes growing in periphery.

#### Graph Two :

This graph shows how  $E$  (defined in previous section) varies with radius. For each ring the average value of  $E$  was determined and plotted. From this graph it can be seen that  $E$  is largest for small values of radius and then steadily decreases as radius increases. This shows that the approximation becomes better as the log-spiral pixels become larger, as expected.

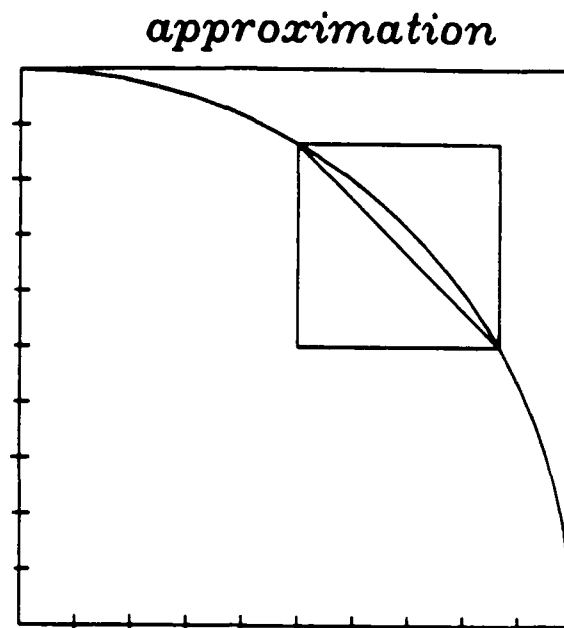


Fig. 6.5 Diagram showing straight line approximation of circular arc across square pixel.

### Graph Three :

Graph Three : This graph shows how error one, error two, and error one + error two vary with angle, ( $0 < \text{angle} < \pi/2$ ). For each ray of pixels the average values of each of these quantities was determined and plotted. It can be seen that the graphs are symmetric about  $\pi/4$ . This is due to the number of pixels per ring being a multiple of four. This causes the computation plane to be symmetric about  $\pi/4$  and, hence, these graphs. The values of error one and error two can be seen to be lowest at angles of 0 and  $\pi/2$ . This is due to the log-spiral pixels sharing a boarder with the square pixels at these angles and, thus, tends to lower the value of these errors.

### Graph Four :

image planes : 1) 600x600 array of square pixels, ( $-5 < x, y < 5$ )

2) 500x500 array of square pixels, "

3) 400x400 array of square pixels, "

4) 333x333 array of square pixels, "

computation plane : log-spiral array of pixels, 36 rings and

36 pixels per ring. Radius of inner ring=0.5,

radius of outer ring = 5.0

This graph shows how the curves E vs. radius vary for square arrays of different sizes. As

expected, the smaller the individual square pixel sizes, the lower E is at a given radius.

#### **Graph Five :**

image plane : 512x512 array of square pixels,  $(-5 < x, y < 5)$

computation planes : log-spiral array of pixels, Radius of inner

ring = 0.5, radius of outer ring = 5.0

- 1) 28 rings, 28 pixels per ring
- 2) 44 rings, 44 pixels per ring
- 3) 52 rings, 52 pixels per ring
- 4) 60 rings, 60 pixels per ring

This graph shows how the curves E vs. radius vary with different computation plane geometries. As expected, the smaller the individual log-spiral pixels the larger E is at a given radius.

The next possible solution to be examined is the case where the camera used is a standard CCD camera with slightly rectangular pixels. A program was written to analyze this case since many of the available CCD cameras actually have rectangular pixels rather than square. The results for this case can be found on graph six.

#### **Graph Six :**

image plane : 458x572 array of rectangular pixels,  $(-5 < x, y < 5)$

computation planes : log-spiral array of pixels, Radius of inner

ring = 0.5, radius of outer ring = 5.0

- 1) 28 rings, 28 pixels per ring
- 2) 44 rings, 44 pixels per ring
- 3) 52 rings, 52 pixels per ring
- 4) 60 rings, 60 pixels per ring

This graph shows how the curves E vs. radius vary as the computation plane geometry is changed. The result was very similar to the result obtained with square pixels. The smaller the individual log-spiral pixels the larger the value of E at a given radius.

The results of this graph show that the effect of the pixels being square or slightly rectangular does not significantly alter the value of E. Thus a camera with either square or slightly rectangular pixels could be used.

The next possible solution to be examined is the case where the camera used is a novel sensor with triangular shaped pixels as shown in the Fig. 6.6. A camera with triangular shaped pixels as shown would be desirable since the pixels could be combined to perfectly form rectangular shaped pixels and could probably be combined to form log-spiral pixels better than rectangular pixels could. In order to examine this a Fortran program was written which calculates the values of error one, error two, and E (as defined before) for the case when the image plane has triangular pixels and the computation plane is log-spiral. The results of these calculations are shown on graph seven, Appendix 6.A.

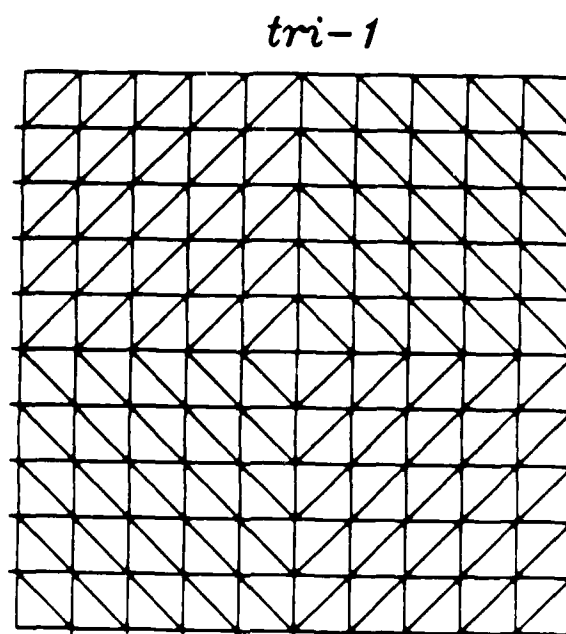


Fig. 6.6 Triangular shaped pixels

**Graph Seven :**

image plane : 362x724 array of triangular shaped pixels

$(-5 < x, y < 5)$

computation planes : log-spiral array of pixels, radius of inner

ring = 0.5, radius of outer ring = 5.0

1) 28 rings, 28 pixels per ring

2) 36 rings, 36 pixels per ring

3) 44 rings, 44 pixels per ring

This graph shows how the curves  $E$  vs. radius vary as the computation plane geometry is varied. As expected, as the size of the individual log-spiral pixels decreased the value of  $E$  at a given radius increased.

In order to determine if there is any advantage in using triangular shaped pixels over rectangular (including square) shaped pixels, graphs eight and nine (Appendix 6 A) were constructed. These graphs combine results obtained for square, rectangular, and triangular shaped pixels.

**Graph Eight :**

image planes : 512x512 array of square pixels  $(-2 < x, y < 2)$

- 2) 458x572 array of rect. pixels       "
- 3) 362x724 array of tri. pixels       "
- computation plane : log-spiral array of pixels, 28 rings and  
28 pixels per ring. Radius of inner ring = 0.5,  
radius of outer ring = 5.0

#### Graph Nine :

- image planes : 1) 512x512 array of square pixels ( $-5 < x, y < 5$ )
- 2) 458x572 array of rect. pixels       "
- 3) 362x724 array of tri. pixels       "
- computation plane : log-spiral array of pixels, 44 rings and  
44 pixels per ring. Radius of inner ring = 0.5,  
radius of outer ring = 5.0

These two graphs show how the curves of E vs. radius compare for three different image plane geometries. The curves for square and rectangular geometries are seen to be very close together. The curves for the triangular geometry had values for E which were for the most part less than the values of E for the other geometries.

Since the image plane geometries were chosen so as to make individual pixel areas the same for all three cases, it would appear that using an image plane which has triangular shaped pixels would better yield a log-spiral representation. The rectangular representation which is obtained by combining two triangular pixels for each rectangular pixel necessarily creates a rectangular representation with pixels twice the size of the individual image plane pixel. Thus, the rectangular representation created would have less resolution than if a camera with rectangular shaped pixels of size equal to the triangular pixels were used to obtain the rectangular representation.

Another possible solution to the dual mapping problem is also analyzed as a matter of curiosity. This solution consists of an image plane with log-log shaped pixels as shown in Fig. 6.7. A camera which had its photosensors laid out in this manner would obtain a log-log representation of images. A log-log representation of images is essentially a rectangular representation with a similar characteristic of log-spiral representations. That is, pixels in the periphery are larger. Thus less pixels are required to describe a given image and algorithms which use this representation should be faster. In order to determine how well log-log shaped pixels combine to form a log-spiral representation, a Fortran program was written which calculates values of error one, error two, and E (as defined before) for the case when the image plane is log-log shaped and the computation plane is log-spiral. The results of these calculations are shown on graph ten, Appendix 6.A.

#### Graph Ten :

- image plane : 200x200 array of log-log shaped pixels, ( $-5 < x, y < 5$ )
- computation plane : log-spiral array of pixels, radius of inner  
ring = 0.5, radius of outer ring = 5.0
- 1) 28 rings, 28 pixels per ring
- 2) 36 rings, 36 pixels per ring
- 3) 44 rings, 44 pixels per ring

*loglog*

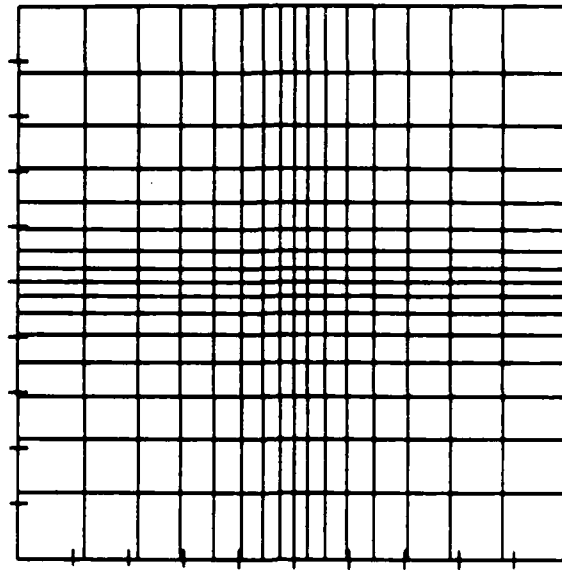


Fig. 6.7 Log-log shaped pixels

This graph shows how the curves  $E$  vs. radius vary as the computation plane geometry is altered. As expected, as the size of the individual log-spiral pixels decreased the value of  $E$  increased for a given radius. The values of  $E$  are significantly higher than those found for similar cases in which square and triangular shaped pixels were used. However, this does not mean that a log-spiral representation obtained from combining log-log shaped pixels would not be adequate. This will depend on the particular application being considered.

## 6.6 A Possible Hardware Implementation of Single-Channel Summing

Figure 6.8 shows a possible hardware implementation of single-channel summing. At the beginning of a pixel-summing cycle, the A register is zeroed, and the count-down and S registers are loaded with the run-length of the next polar pixel. As each rectangular element is shifted in, it is passed through the A/D converter and summed in ALU1. The results of each addition are placed in register A so that a new element can be added to the sum. When the count-down register reaches zero, the output of the Op amp (which has been summing the pixel data) is fed to the A/D converter. The ALU divides that digitized sum by the run-length for that pixel, thereby performing the averaging; the Op amp dump switch is activated, zeroing the output, and the next run-length is loaded.

Figure 6.9 shows a possible hardware implementation of digital summing. At the beginning of a pixel-summing cycle, the A register is zeroed, and the count-down and S registers are loaded with the run-length of the next polar pixel. As each rectangular element is shifted in, it is passed through the A/D converter and summed in ALU1. The results of each addition are placed in register A so that a new element can be added to the sum. When the count-down register reaches zero, the output of the Op amp (which has been summing the pixel data) is fed to the A/D converter. The ALU divides that digitized sum by the run-length for that pixel, thereby performing the averaging; the Op amp dump switch is activated, zeroing the output, and the next run-length is loaded.

### 6.6.1 Operation of the analog summing arrangement (Fig 6.8)

- 1) The count-down and S registers are loaded with the run-length of the next polar pixel, being shifted out of the CCD.
- 2) As data is shifted out of the sensor array, and into the summing hardware, the count-down register is decremented. When the register reaches zero, the output of the Op amp (which has been summing the pixel data) is fed to the A/D converter. The ALU divides that digitized sum by the run-length for that pixel, thereby performing the averaging; the Op amp dump switch is activated, zeroing the output, and the next run-length is loaded.
- 3) Operation continues until all pixels in the current frame have been read.

### 6.6.2 Operation of the digital summing arrangement (Fig 6.9)

- 1) At the beginning of a pixel-summing cycle, the A register is zeroed, and the count-down and S registers are loaded with the run-length of the next polar pixel.
- 2) As each rectangular element is shifted in, it is passed through the A/D converter and summed in ALU1. The results of each addition are placed in register A so that a



Figure 6.8 Analog pixel summing arrangement.

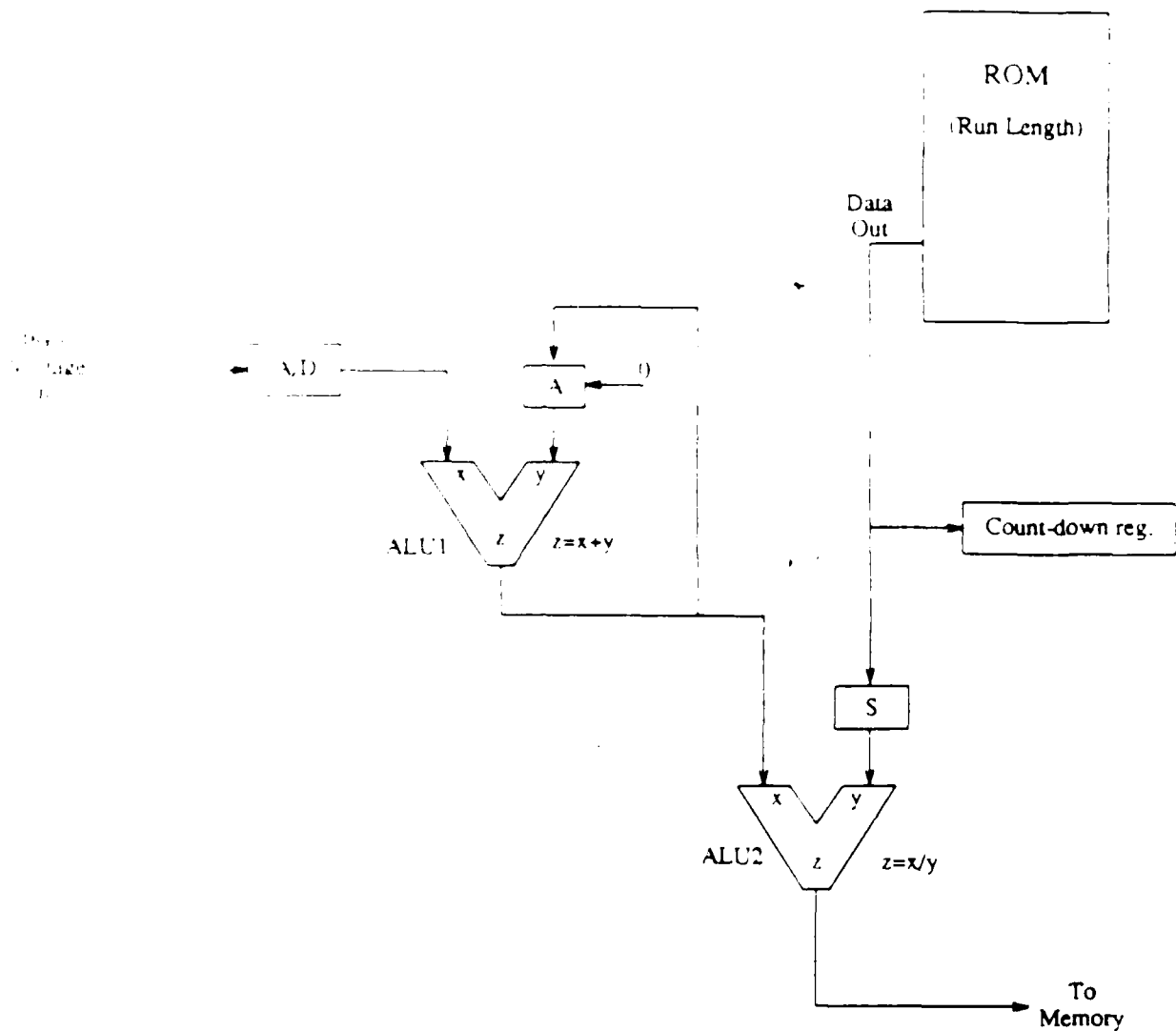


Figure 6.9. Digital pixel-summing arrangement.

- cumulative sum may be obtained. Also, the count-down register is decremented.
- 3) When the count-down register indicates that the last element has been summed, the total is divided (in ALU2) by the run-length for that pixel (stored in the S register), thereby forming the average pixel intensity.
  - 4) The cycle continues until the entire frame has been read.

A simple example of this operation follows. We will use the second approach (digital summing) and detail the operation in a clocked fashion. Assume a single- or multi-

phase clocking arrangement in which, during each complete clock cycle, a single rectangular element is shifted out of the sensor and both ALUs are able to perform their required functions (if called upon to do so). We also assume that the hardware requires one extra clock cycle to perform the set-up operations necessary prior to summing the elements which form a particular pixel. For this example, the run-lengths of the first three polar pixels to be extracted (all that we will consider at this time) are 4, 17, and 32 (completely arbitrary at this point since we do not know the layout of the polar pixels). Simulation is as follows (clock cycle followed by functions performed):

- 1) Zero ROM address. Load count-down and S registers with run-length of first polar pixel to be shifted out (4). Clear the A register.
- 2) Shift first rectangular element into A/D converter (flash converter) and add output to value in the A register (0). Store result in the A register. Decrement count-down register. Count-down register does not equal zero (yet) so continue.
- 3) Shift next rectangular element into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register.
- 4) Same as 3).
- 5) Same as 3). However, now the count-down register is zero. Therefore, divide the output of ALU1 by the contents of the S register (the run-length) in ALU2 to form the average pixel intensity. Store this in memory.
- 6) Increment the ROM address to point to the next run-length. Load the count-down and S registers with the next run-length (17). Clear the A register.
- 7) Shift first rectangular element of the next polar pixel into the A/D converter and add output to value in the A register (0). Store result in the A register. Decrement count-down register.
- 8) Shift next rectangular element into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register.

continue ...

- 23) Shift next rectangular element (17<sup>th</sup> in this pixel) into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register. Count-down register is now zero. Therefore, divide the output of ALU1 by the contents of the S register (17) in ALU2 to form the average pixel intensity. Store this in memory.
- 24) Increment the ROM address to point to the next run-length. Load the count-down and S registers with the next run-length (32). Clear the A register.
- 25) Shift first rectangular element of the next polar pixel into the A/D converter and add output to value in the A register (0). Store result in the A register. Decrement count-down register.

26. Shift next rectangular element into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register.

Continue

26. Shift next rectangular element (1) in this pixel into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register. Count-down register is now zero. Therefore, multiply the output of ALU 1 by the contents of the S register (12) in ALU 2 to form the average pixel intensity. Store this in memory.
27. Increment the ROM address to point to the next run length. Load the count-down and S registers with the next run length. Clear the A register.

Continue until entire array is read

This algorithm is very systematic and easily implemented, as shown above.

Since pixels on the innermost ring of the polar arrangement have the smallest area, they will be formed from the smallest number of rectangular elements. Consequently, all timing considerations and error analysis must be performed using these pixels. That this is so for error analysis is evident from an averaging point of view. Since the polar pixel's intensity will be the average intensity of the rectangular elements whose centers lie within its border, the smaller the area of the polar pixel, the greater will be the error (or error variance) of the estimated intensity. As for timing considerations, since one clock cycle is wasted in preparation for each polar pixel (preparing the hardware to sum and average the incoming rectangular elements), the smaller the polar pixel area, the greater will be the clock cycle overhead expended. Hence, the dependence of timing considerations on the innermost ring of polar pixels (smallest polar pixels) is justified.

The subject of error analysis is one which must certainly be discussed since there will never be a perfect fit of rectangular elements into any given polar pixel. It is this area which we will now address. All analysis will be performed using an arc-of-rings configuration for the logarithmic-spiral sensor. The CCD sensor will consist of  $N \times N$  square light-sensitive elements (the term rectangular, when used, refers to the overall sensor configuration rather than actual pixel dimensions).

### 6.7 Error Analysis

As seen in Fig. 6.4, there is never a perfect fit of rectangular elements into polar pixels. We have chosen to assign a given rectangular element to a polar pixel if its center is within the defined boundary of that pixel. So, not only will there be error in the area covered by the rectangular elements, but some of the intensity information attributed to a given pixel will be from outside its boundary. Furthermore, any actual error analysis must necessarily rely on the specific arrangement and resolution of the polar pixels and rectangular elements. That is, in a given rectangular array of elements, the number of



inner radius of the innermost ring,  $r_0$ . The inner boundary of the innermost ring is  $r_0$  and the outer boundary is given by

$$R_0 = r_0 + \frac{A}{2\pi r_0}$$

where there are  $N_0$  rings with the outer ring of pixels bounded by a radius of  $R_0$ . The total area of all  $N_0$  adjacent rings of pixels is given by

$$A_0 = \sum_{i=0}^{N_0-1} A_i$$

where we calculate  $r_i$  from this

In order for the average error to converge to zero, for an inner ring containing  $N_i$  pixels of area  $A_i$  we require that there be  $\alpha N_i$  polar pixels containing  $(1-\alpha)A_i$  rectangular elements and  $(1-\alpha)N_i$  polar pixels containing  $(1-\alpha)A_i$  rectangular elements. For then the average area is exactly  $A_i$  and the average error is zero. We can then form the sampled variance of the error variable as

$$\sigma^2 = \frac{1}{N_0-1} \sum_{i=0}^{N_0-1} v_i^2$$

where  $v_i$  is the normalized pixel area error

$$v_i = \frac{A_i}{A} \left( \frac{\alpha}{1-\alpha} \right) \text{ or } \frac{1-\alpha}{\alpha} \left( \frac{A_i}{A} \right)$$

Therefore, the sampled variance is given by

$$\sigma^2 = \frac{N_p(1-\alpha)}{N_p-1} \left( \frac{\alpha}{A} \right)^2 + \frac{N_p(1-\alpha)}{N_p-1} \left( \frac{1-\alpha}{A} \right)^2 + \frac{N_p(1-\alpha)}{N_p-1} \left( \frac{2-\alpha}{A} \right)^2$$

If we decide to include those error events which are between one and two pixels away from the actual polar pixel area (termed "double" errors), we must, once again assume some distribution of these events so that the average area converges to the actual pixel area. There are too many variables (the problem is under-constrained) so we must constrain some so that a unique solution is possible. Therefore, we will assume that the proportion of errors of magnitude  $(1-\alpha)$  to those of magnitude  $(2-\alpha)$  is such that their average is also zero. Further, we can assume that these "double" errors account for a portion  $\beta$ , of all error events. An accounting of the quantity and magnitude of each of these error events is given below (per  $N_p$  polar pixels, error normalized).

<u>error magnitude</u>	<u>quantity</u>
$\left( \frac{\alpha}{A} \right)$	$(1-\alpha)(1-\beta)N_p$
$\left( \frac{1-\alpha}{A} \right)$	$\alpha(1-\beta)N_p$
$\left( \frac{-(\alpha+1)}{A} \right)$	$\frac{(2-\alpha)}{3} \beta N_p$
$\left( \frac{2-\alpha}{A} \right)$	$\frac{(1+\alpha)}{3} \beta N_p$
$0 \leq \alpha \leq 1$	
$0 \leq \beta \leq 1$	

The sampled variance then becomes

$$\sigma^2 = \frac{N_p}{(N_p-1)A^2} \left\{ (1-\beta)\alpha(1-\alpha) + \beta(2+\alpha-\alpha^2) \right\}$$

For  $\beta=0$ , this reduces to the former case in which we considered only "single error" events. For the worst case,  $\beta=1$ ,  $\alpha=.5$ , the approximate sampled variance is

$$\sigma^2 = \frac{2.25 N_p}{(N_p - 1) A^2}$$

which is nine times larger than the worst case ( $\alpha=.5$ ) for single error events only.

Choosing a maximum allowable variance of, say, 5%, Figs. 6.10 and 6.11 show how the number of rings,  $N_r$ , relates to the number of polar pixels per ring,  $N_p$ , for arrays of size  $512 \times 512$  and  $1024 \times 1024$  pixels. Plots of  $\beta=0$  and  $\beta=.25$  are shown. Increasing either  $N_r$  or  $N_p$  will increase the error variance -- an adverse effect. Figure 6.12 shows how the area of the pixels on the innermost ring of the polar array relate to the error variance.

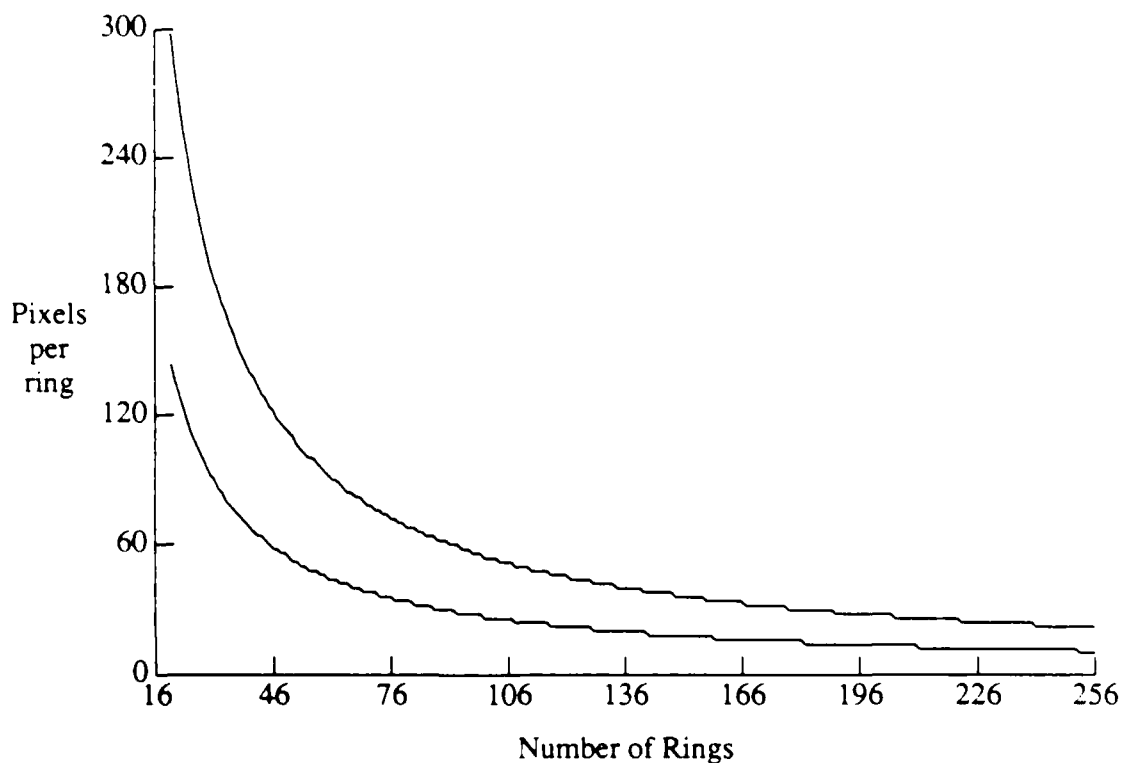


Figure 6.10. Pixels/ring vs. Number of rings for  
Error Variance = .05 (512 x 512 pixels)

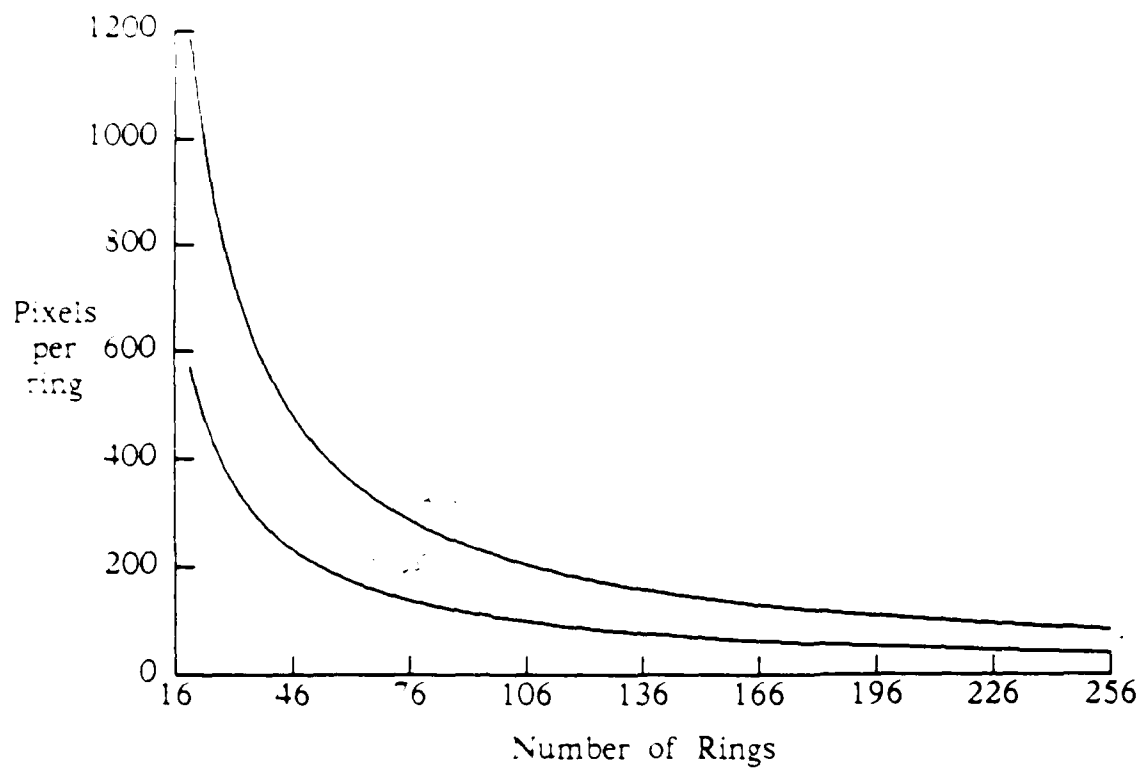


Fig. 6.11 Pixels/ring vs. Number of rings for  
Error Variance = .05 (1024 x 1024 pixels)

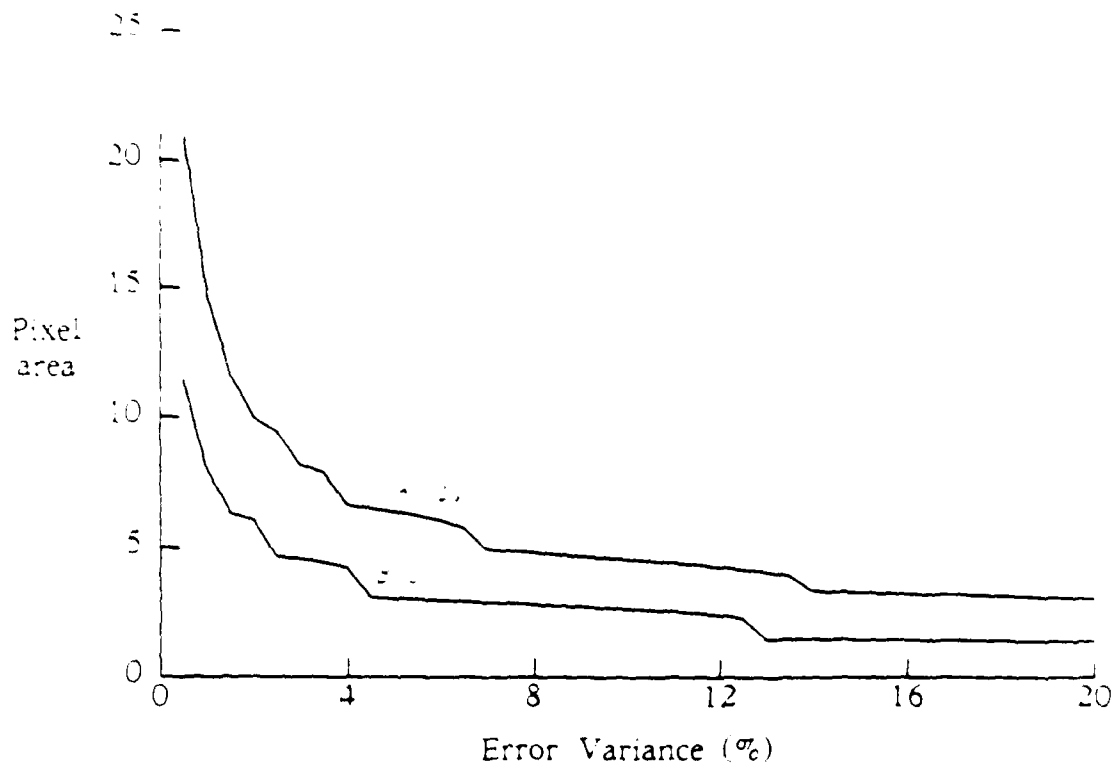


Fig. 6.12 Minimum Pixel Area vs. Error Variance (%).

## 7 DEPTH COMPUTATION FROM OPTIC FLOW

### 7.1 Introduction

Due to the physical constraints of sensor geometry, the projection of the three-dimensional (3D) spatial structure of the environment onto the 2D image projection of systems with a single sensor is severely degraded. The only system that could support 3D computation is optic flow generated by the motion of the observer. Object motion alone is insufficient since it is ambiguous as to whether the object or other depth cues such as size, occlusion, or other perspective cues might provide an alternative and provide inaccurate or misleading information. Thus, the computation of 3D space from optic flow induced by sensor motion also provides information on the MDS function of biological visual systems and much information on the forms of depth perception, planning and control.

Under the condition of EM where the line of sight (LOS) corresponds with the direction of linear motion, the visual images of moving environmental objects undergo accelerated radial expansion from a central point. This point, termed the focus of expansion (FOE), corresponds to the extension of the line of motion in the opposite direction. Thus, under EM, world objects visually move along arcs which extend from the FOE to the focus of contraction (FOC).

Virtually all of the work done on the computation of 3D space from EM has been limited to this case where the LOS corresponds with the FOE. In most visual systems, however, the visual field (VF) encompasses only a limited portion of the 360° view. Lateral rotation of the visual sensor in a moving system thus results in the deviation of the LOS from the FOE. Geometrically, objects continue to move along arcs from the FOE to the FOC. This rotation, however, results in the transformation of motion in the VF from radial expansion to lateral translation as the angle between the LOS and the FOE approaches 90°. Computation of 3D space is more difficult under these conditions.

In the following sections we will discuss research on obtaining and utilizing the information inherent in optic flow for the computation of depth and structure in the VF. An algorithm for the determination of the flow field will also be discussed along with an examination of the problems which arise in such computations. Possible physiological correlates of optic flow computations will be presented next. Finally, a new approach to the problem using the BVS and spherical symmetry will be examined.

### 7.2 Overview of Current Research

Work on the determination of object structure and depth from EM has developed rapidly over the last decade. Various schemes have been proposed for the determination of optical flow and the extractor of information from the flow field. A general criticism of most current work is that the proposed schemes are computationally intensive and therefore of limited usefulness in real-time applications. Rather than review this work in detail, however, we will attempt to provide a general understanding of the current paradigm by examining the work of selected researchers.

Prazdny [2-4] has made significant contributions to this field. He analyzes optic flow in terms of the projection of a six parameter transformation (three translational and three rotational parameters) in a Cartesian coordinate system. Prazdny has

Ady [5] has characterized the the optic flow [3] and shown that, though the instantaneous velocity vectors specify only the translational component of EM (not rotational), the flow does specify the relative depth, from the observer, of objects in the VF. This works is based on three non-trivial assumptions:

- (1) instantaneous velocity vectors are available at all retinal locations,
- (2) retinal surfaces and objects are rigid
- (3) the position of points on the retina with respect to the center of gaze is known.

Instantaneous velocity vectors depict the direction and magnitude of motion for image elements at each retinal location. The determination of these vectors is, of itself, a non-trivial problem.

Ady [5] has proposed a scheme for determining both the structure of 3D space and the parameters of 3D motion of objects in the VF. He utilizes a two stage process. The first stage determines the optical flow field and the second stage interprets this field, obtaining information about the structure and motion parameters.

Ady points out that optical flow fields are noisy in the sense that velocity vectors may be locally uncorrelated and that this lead to a loss or corruption of the available information. Another problem arises if there are multiple objects in the field of view. This can result in occlusion which leads to singularities (discontinuities) in the flow field. Algorithms for interpreting optical flow must be robust enough to handle these sorts of problems.

Ady's approach first partitions the flow field into connected segments. These segments are then merged under the hypothesis that segments with similar characteristics result from the same rigidly moving object.

The algorithm searches for 3D motion parameters which satisfy all segments within a merged set. Relative depth is then obtained from these parameters. This scheme is relatively impervious to the presence of noise and, in addition, can handle cases where objects in the view are moving independently (in addition to the EM).

Jain and his colleagues [6] have also examined the computation of the relative depth from EM. Their work is of particular interest in that they utilize a polar coordinate sensor geometry rather than the common rectilinear geometry used by other workers. When mapped to a computation plane via a conformal complex logarithmic mapping optical flow can be analyzed as a single dimension ( $z$ ) motion. This depends, however, on the LOS being identical to the FOE, thus being a radial flow. If the LOS diverges only slightly from the FOE, the method fails due to distortions in the optical flow pattern.

### 7.3 Computation of Optical Flow

The work discussed above assumes that fairly accurate flow field information is available. Obtaining this information, however, is a significant problem. A number of schemes are available for computing determination of correspondences for prominent features of the image across frames. The main drawback to this approach is that general solutions to the correspondence problem do not exist. Thus, these methods tend to be ad hoc or domain specific.

Buxton & Buxton [7] have presented an approach which assumes that changes in the intensity function are due only to the motion induced by the EM. The optic flow data

NO-A191 157

A SENSOR WITH BIOLOGICAL PREPROCESSING FEATURES(U)  
VIRGINIA UNIV CHARLOTTESVILLE DEPT OF ELECTRICAL  
ENGINEERING R M INIGO ET AL 18 DEC 87

272

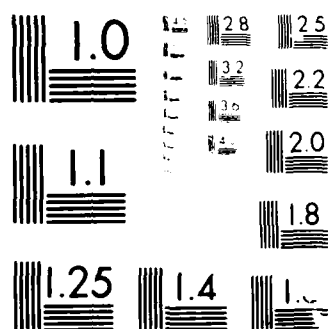
UNCLASSIFIED

UVA/525679/EE88/101 AFOSR-TR-88-0048

F/G 12/9

NL





MICROCOPY RESOLUTION TEST CHART  
NBS 1010-A

can then be obtained by a low-level spatio-temporal difference of Gaussians (DOG) filter to compute the location of moving edges within the input. Our group has used this filter previously and shown its usefulness [8]. The DOG filter determines zero-crossings (second derivative of the intensity factor) which are equivalent to edge information. Optic flow is obtained from the zero-crossing data through a least squares procedure. This results in the computation of what Buxton and Buxton term vernier velocities.

A major difficulty with this approach is that it is computationally intensive, requiring the performance of approximately 50,000 convolutions for each 128 X 128 frame pair. Significant savings in computation time, however, can be achieved through the use of parallel processing.

#### **7.4 Physiological Correlates of Optic Flow Computation**

The importance of 3D computation from optic flow for biological organisms indicates that the nervous system should contain specialized mechanisms for performing this function. The available evidence indicates that these mechanisms may involve multiple visual areas of the cortex, primarily area 18 and the inferior parietal visual area.

An examination of the distribution of velocity sensitive cells in area 18 indicates that within 10 degrees of the area central is the major proportion of cells are velocity tuned [9]. These cells are bandpass for velocity and generally are directionally sensitive. These properties are well suited for the detection and tracking of object motion. Beyond 10 degrees eccentricity, the proportion of velocity tuned cells decreases rapidly and the major proportion of cells are velocity highpass, with orientation but not direction selectivity [9]. These cells show responses that are linearly increasing with the log of the velocity. These properties are well suited for the computation of optic flow.

Parietal visual neurons (PVNs) have been extensively studied by Mountcastle and his colleagues [10]. PVNs have large, bilateral receptive fields and are responsive to motion but apparently are not velocity tuned. PVNs have a complex "opponent radial" receptive field organization [10]. They are sensitive to motion either toward or away from the center of their receptive fields along some preferred axis. In addition, these cells can show differential responses as a function of whether or not the motion crosses the center of the field.

PVNs are excellent candidates for the performance of higher level processing of 3D space. Whereas area neurons may function in the determination of optic flow, PVNs, which are afferently connected to area 18, may utilize this information in the construction and maintenance of a 3D model of the environment. This idea is strengthened by the fact that parietal lesions can result in severe impairments of spacial processing and visually guided behavior [10].

#### **7.5 An Approach to 3D Computation from EM**

It was pointed out in previous discussions that virtually all schemes for determining depth from EM depend on an alignment of the line of sight and the focus of expansion. As the LOS deviates from the FOE, the radial flow of the image becomes progressively distorted toward lateral translation over the visual field. This distortion complicates the determination of the optic flow, so that some schemes [e.g. 3D] incorporate methods for locating the FOE within the flow. We are developing an approach which eliminates this

problem and allows the determination of depth regardless of the LOS/FOE relation.

An important aspect of our approach is the determination of a 3D reference frame. Most previous work computes 3D motion relative to a set of predefined 3D coordinates. We claim that a coordinate system can and should be determined on the basis of the optic flow itself. This provides for greater adaptability of the visual system and allows the recalibration of the 3D model of the environment following disorientation.

Our approach involves four primary components.

1) Flow Field Detection. The BVS sensor will be used for the detection of image motion. Two possible schemes for the detection of motion are i) the use of the spatio-temporal DOG filter or ii) an adaptation of the correlation method developed by Narathong [see section].

2) Mapping to Computation Plane. The use of the BVS sensor allows the use of the conformal complex logarithmic mapping to the computation plane. We propose that the computation "plane" be conceptually understood as the surface of a sphere, defined by three parameters,  $l$ ,  $-$  and  $r$ . This sphere is oriented such that the direction of motion is always aligned with its polar axis. Then the parameter  $l$  corresponds with lines of latitude,  $-$  corresponds with the longitude lines and  $r$  represents the radius of the sphere. This sphere can be represented as a two dimensional ( $l$  and  $-$ ) surface where  $0 < l < \pi$  and  $0 < - < 2\pi$ .

3) Organization of the Flow Field. The spherical computation plane reflects the invariant geometry of the flow field in that, under EM, all world objects visually move along arcs from the FOE to the FOC. Thus, since the polar axis of the sphere is aligned with the direction of sensor motion, all image motion can be analyzed in terms of motion along a single dimension ( $l$ ). This, however, depends on the proper pattern of activation of the computation plane. Given a limited VF, it must be mapped to the computation plane across an area that is congruent with the orientation of the VF relative to the direction of the EM. Since individual points are ambiguous with respect to this function, this organization will depend on the relation between velocities the entire VF. This entire issue is eliminated, however, if we allow the receptor surface to be a sphere as well. In this case, the receptor and computation planes are always congruent. This is very similar to the situation found in insect vision, e.g., dragonflies, or other animals with 360 degree VFs.

4) Computation of Depth. Given the congruence between the sensor array output and the computation plane, activity levels at each point in this plane will be proportional to the distance of some surface element from the sensor along the line of sight represented by that location in the computation plane.

These four stages, motion detection, mapping, flow field organization and depth computation provide the basis for the determination of 3D spatial structure and motion from EM. This system is relatively simple in its basic construction due to the advantages of the BVS and log conformal mapping for the determination of radial motion.

### References

- [1] Gibson, J.J. (1966) *The senses Considered as Perceptual Systems*. Boston, MA: Houghton-Mifflin.
- [2] Prazdny, K. (1980) Egomotion and relative depth map from optical flow. *Biological Cybernetics*, 36, 87-102.
- [3] Prazdny, K. (1981) Determining the instantaneous direction of motion from optical flow generated by a curvilinearly moving observer. *Computer Graphics and Image Processing*, 17, 238-248.
- [4] Prazdny, K. (1983) On the information in optical flows. *Computer Vision, Graphics, and Image Processing*, 22, 239-259.
- [5] Adiv, G. (1985) Determining three-dimensional motion and structure from optical flow generated by moving objects. *IEEE Transactions on PAMI*, PAMI-7, 384-401.
- [6] Jain, R., Bartlett, S. L. and O'Brien, N. (1987) Motion stereo using eog-motion complex logarithmic mapping. *IEEE Transactions on PAMI*, PAMI-9, 356-369.
- [7] Buxton, B. F. and Buxton, H. (1984) Computation of optic flow from the motion of edge features in image sequences. *Image and Vision Computing*, 2, 59-75.
- [8] Hsin, C., "Edge and Motion Detection Using Peripheral Human Visual System," Master of Science Thesis, Electrical Eng. Dept., University of Virginia, Charlottesville, VA, March 1987.
- [9] Orban, G. A. (1984) *Neuronal Operations in the Visual Cortex*. New York: Springer-Verlag.
- [10] Mountcastle, V.B., Motter, B.C., Steinmetz, M.A. and Duffy, C.J. (1984) Looking and seeing: The visual functions of the parietal lobe. In G. M. Edelman, W. E. Gall and W. M. Cowan (eds.), *Dynamic Aspects of Neocortical Function*. New York: John Wiley & Sons.

## 8. VESTIBULAR-OCULAR MOTION FOR TARGET TRACKING

Models of human eye movement have been studied. There are many features displayed in the abilities of biological systems which could be exploited in the design of a biological visual sensor. Among these seem to be the accuracy with which animals are able to track target motion and, in particular, the periodic movement of targets. It should be mentioned at the outset that, though accurate models of this behavior appear to have been made, their usefulness is doubtful. The most recent of these models studied, which incorporates features of its predecessors, relies on methods of estimation of target motion from past observations. Whereas this model performs well in the tracking of regular target motion, the advantages seem small. The models will be discussed nonetheless.

Young and Stark [1] proposed a model of human smooth pursuit and saccadic eye movements as a sampled-data control system. No attempt was made to handle regular periodic motion in any fashion other than that used for non-periodic target movements. The eye movement signals for these two types of motion are generated in parallel and are additive. Smooth pursuit motion is stimulated by the retina velocity error between target and eye movements, whereas saccadic motion is generated by retinal position error. Their model incorporated these known features, even delaying the saccades by one sampling period ( $\approx 200\text{ms}$ ) from the time the position error was measured, in agreement with experimental results. This model simulated actual eye movement to non-periodic motion very well.

Eckmiller [2] discussed Neural Control of Foveal Pursuit and Saccadic eye movements. He was concerned with the synaptic paths and signal generation methods which characterize the primate oculomotor system and its ability to pursue moving visual targets or direct their optical axis towards briefly presented stationary targets. His proposed model incorporated a sequence of three major functional areas. They are the spatiotemporal translator, the motor program generator and the neural integrator blocks.

Spatiotemporal Translation is concerned with the transformation of spatial position error information of retinal signals into a smooth pursuit velocity error. The article placed major emphasis on defining the "neuroanatomical architecture" (which defines the connections between input neurons, representing specific retinal locations, and output neurons of the Spatiotemporal Translator) as vital to the realization of the Spatiotemporal Translator.

The spatiotemporal Translator provides signals to the Motor Program Generator (MPG), the second block of the signal pathway. Eckmiller sites the MPG's as the source of time courses of neural activity. That is, oculomotor activity is in response to signals generated in the MPG's, regardless of current input from the Spatiotemporal Translator block. (The Spatiotemporal Translator supplies signals to the MPG for interpretation.) It is in the MPG's that models of regular waveforms are generated so that the eyes can follow targets (which follow such waveforms) without any latency or error. "Very little is known about the the neural realization of different motor program generators," and so it is in this area that much research needs to be done.

Eckmiller concludes that further simulation of these models requires the resolution of fundamental questions concerning biological systems. Specifically, how the motor program generator stores and updates different motor programs for smooth pursuit

movement is not known. Further, the mathematical algorithm of the neural predictor mechanism must be researched.

A third model, by Bahill and McDonald [3], called the "Target Selective Adaptive control" (TSAC) model, aims to describe human eye movement in response to periodic target motions. In addition to the smooth pursuit and saccadic branches, this model incorporates a "TSAC" branch to estimate target motion and provide the eye with a suitably time-advanced version of the motion so that, after the predictable delays of the eye mechanism have occurred, eye movement stays locked on to target position. This model was simulated not with a "library" of acceptable input waveforms, as the authors preferred, but with a finite differences method of target motion estimation. It was able to formulate an equation describing target motion using  $n+1$  samples for an  $n^{\text{th}}$  order input waveform. Problems occurred if the periodic waveforms were corrupted with even small levels of noise.

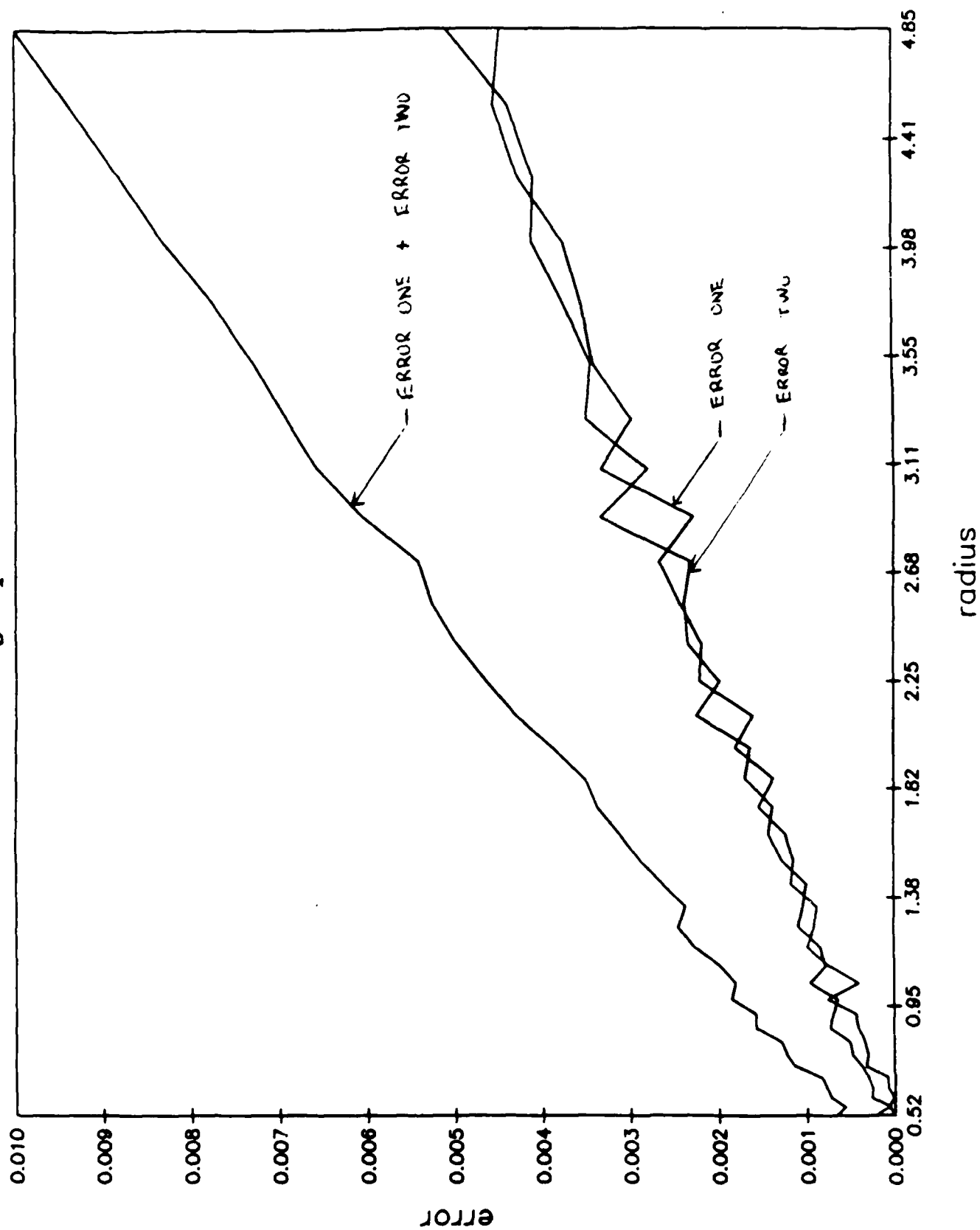
As mentioned above, it is not deemed necessary to incorporate these estimation techniques into the BVS as their gains are marginal. Conventional position control methods should provide adequate sensor orientation and are not fraught with the noise problem to which we have alluded.

#### REFERENCES

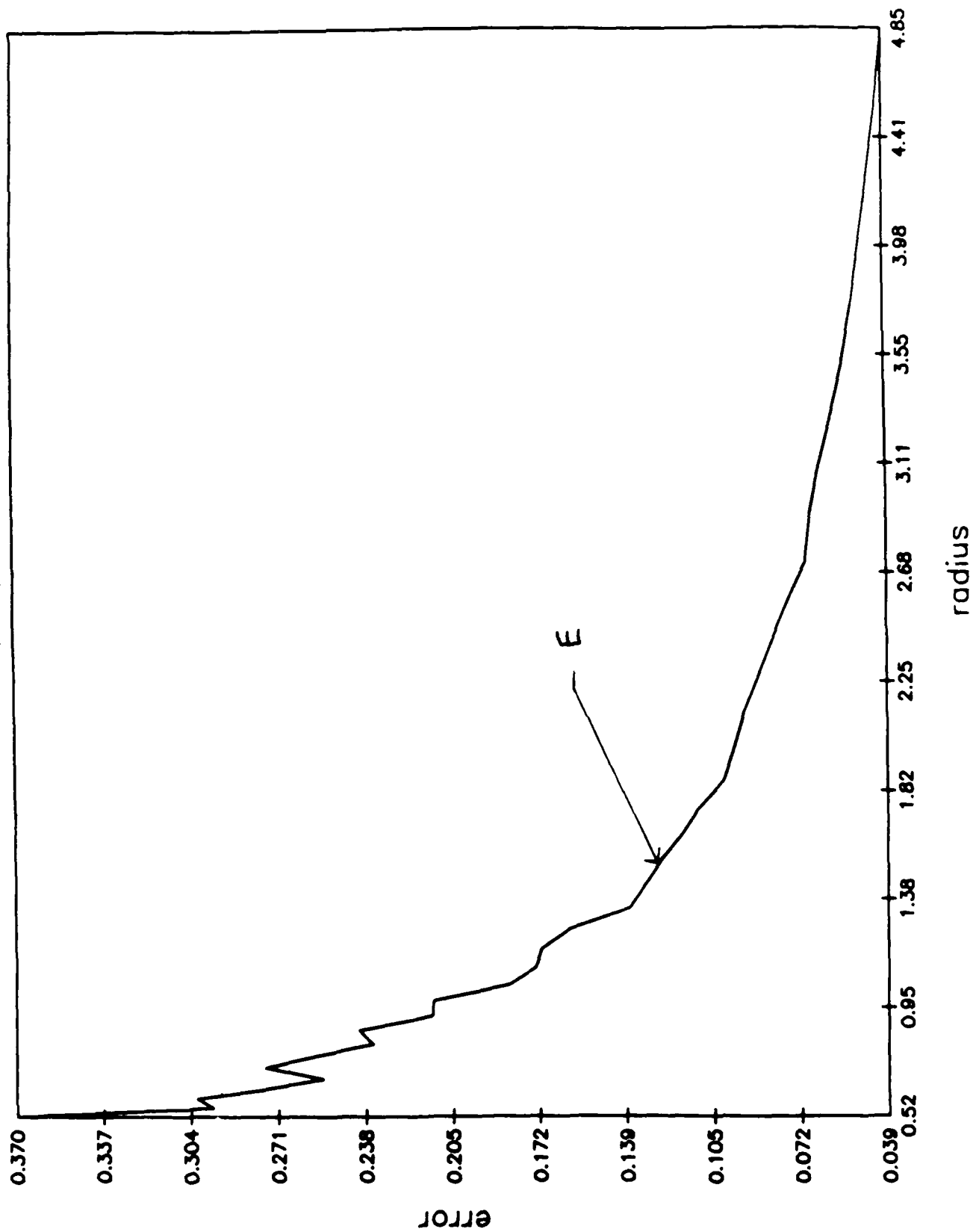
- [1] Young, L.R., and Stark, L., "Variable Feedback Experiments Testing a Sampled-Data Model for Eye Tracking Movements," IEEE Transactions on Human Factors in Engineering, Sept. 1963, pp. 38-51.
- [2] Eckmiller, R. "Neural Control of Foveal Pursuit Versus Saccadic Eye Movements in Primates -- Single-Unit Data and Models," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, Number 5, September/October 1983, pp. 980-988.
- [3] Bahill, A.T., and McDonald, J.D., "Model Emulates Human Smooth Pursuit System Producing Zero-Latency Target Tracking," Biological Cybernetics, Vol. 48, 1983, pp. 213-222.

**APPENDIX 6.A**  
**ERROR GRAPHS**

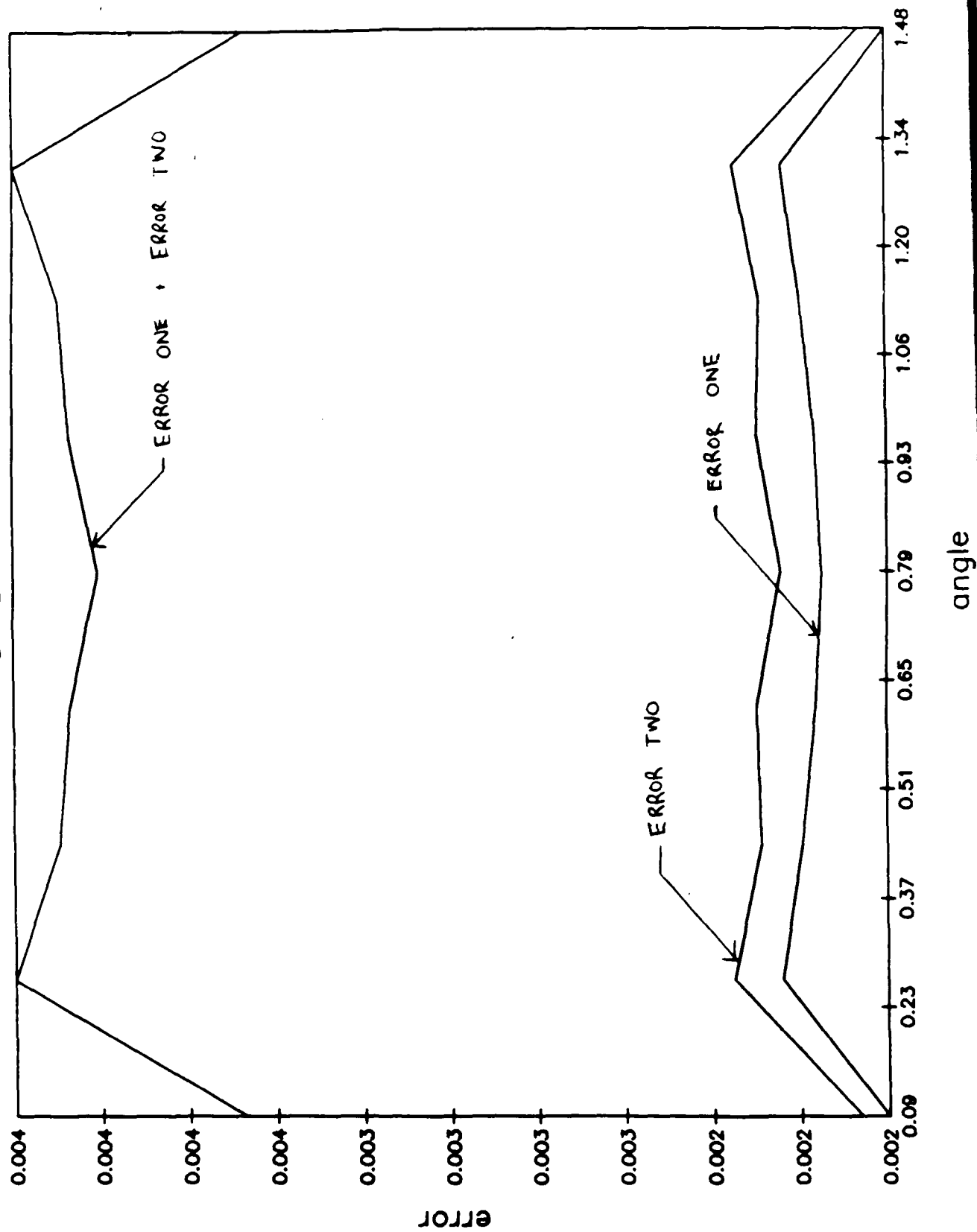
graph one



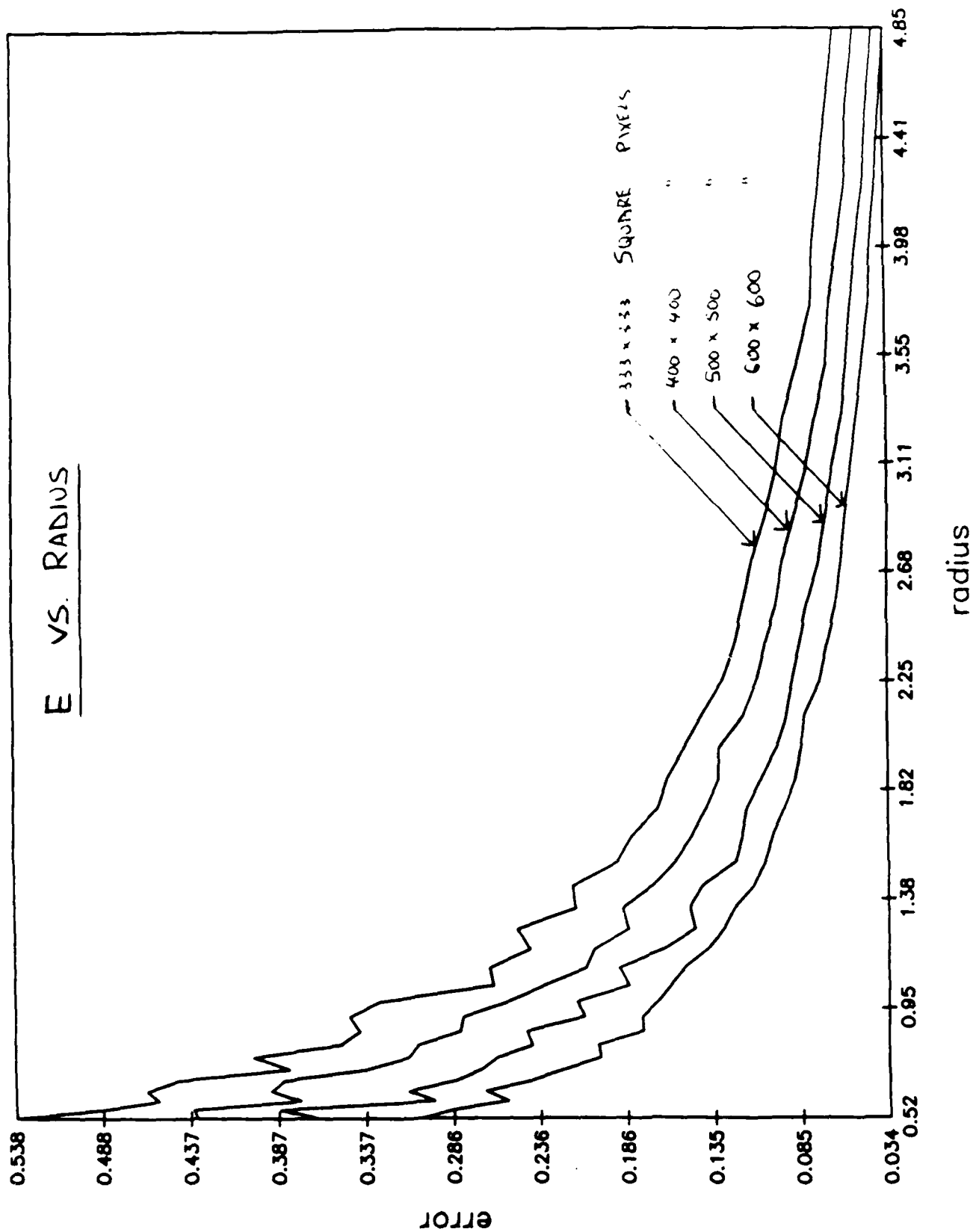
graph two



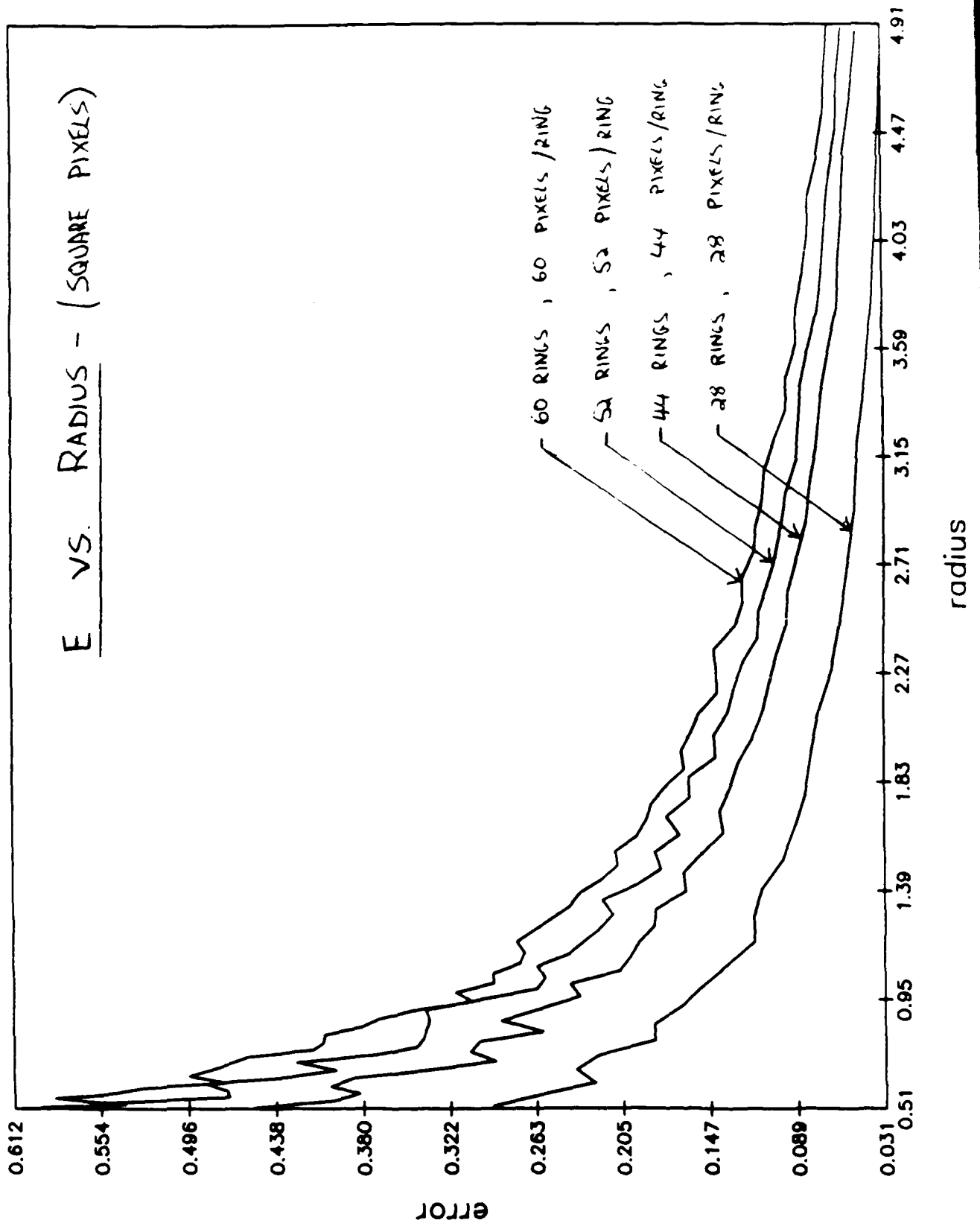
graph three



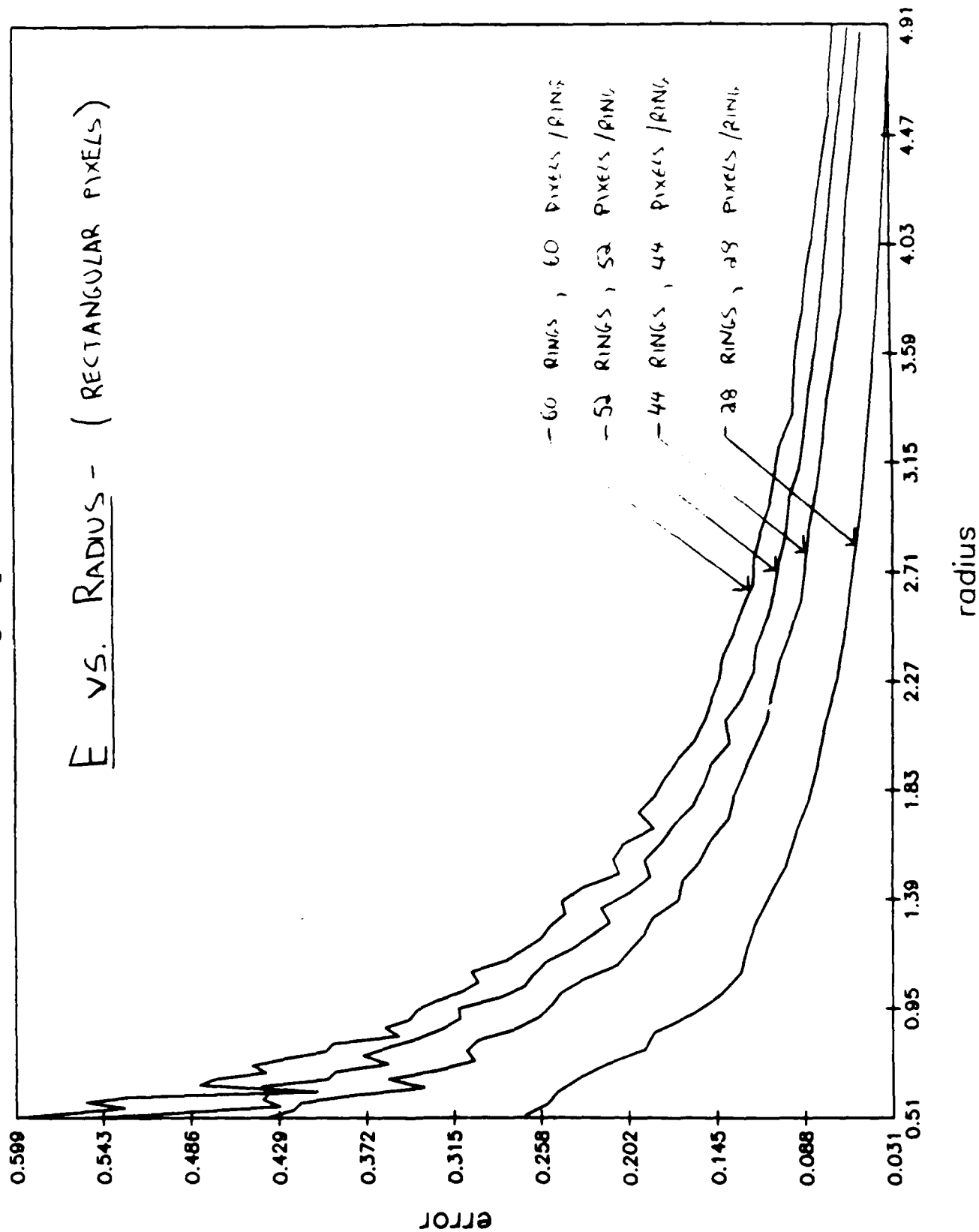
graph four



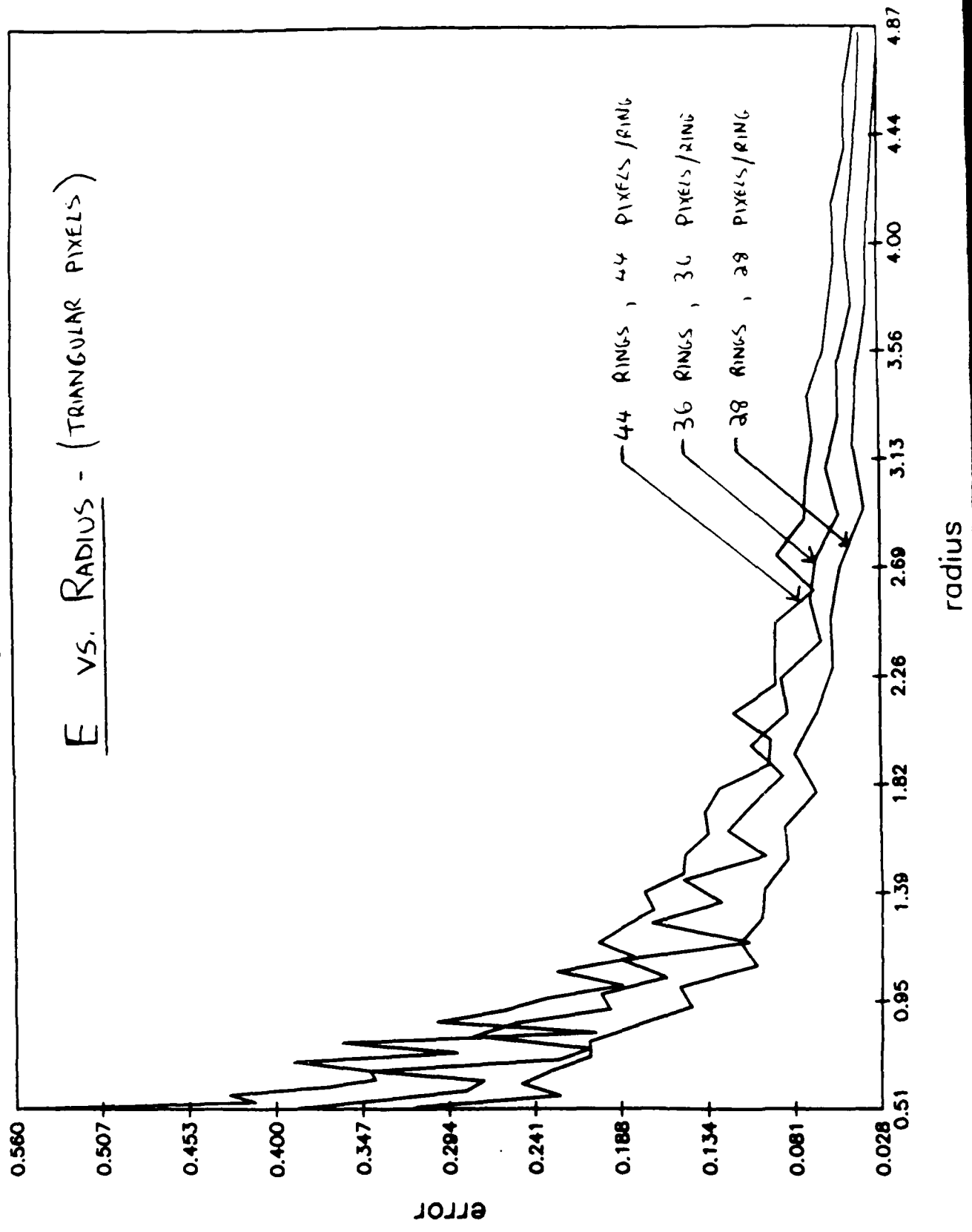
graph five



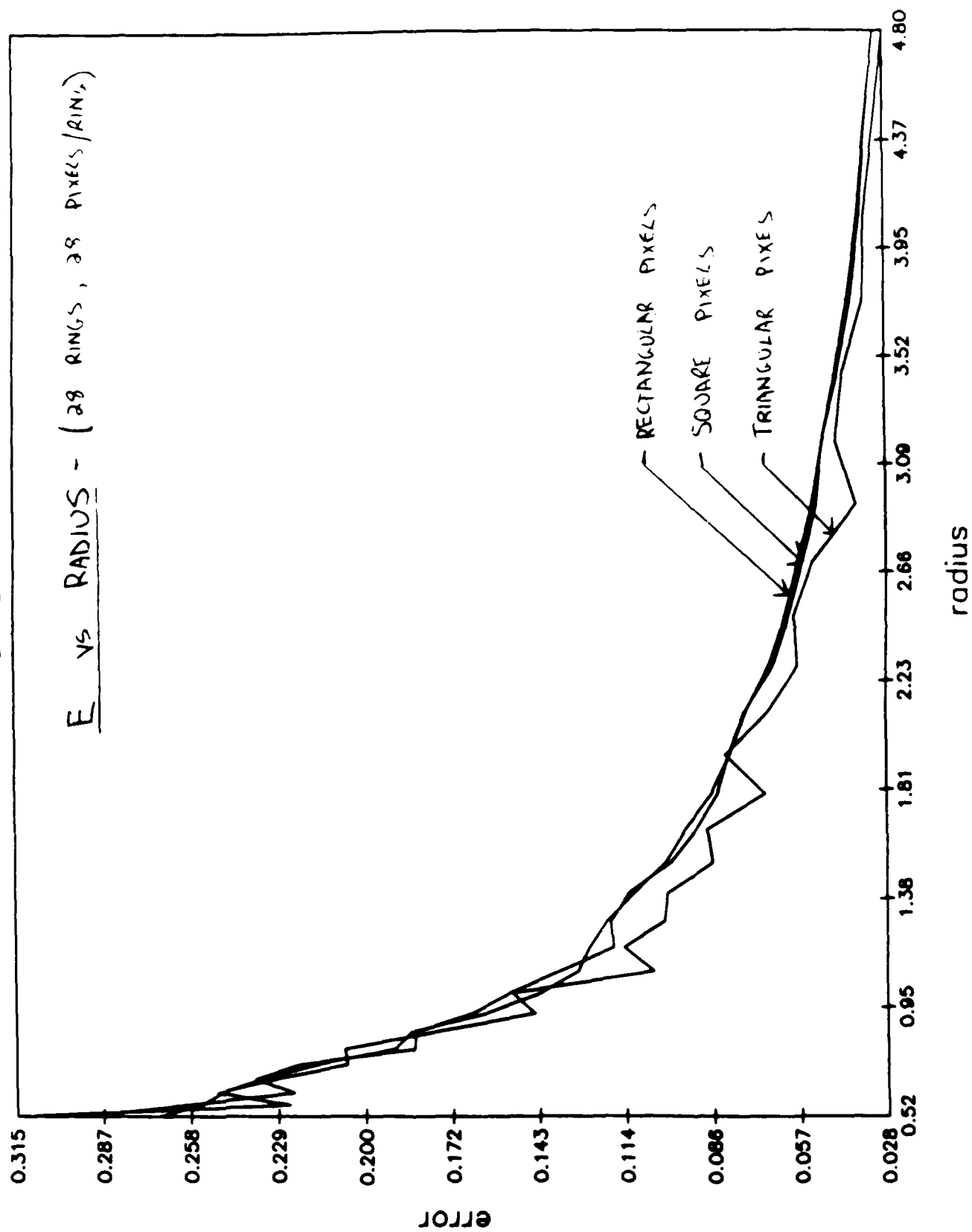
graph six



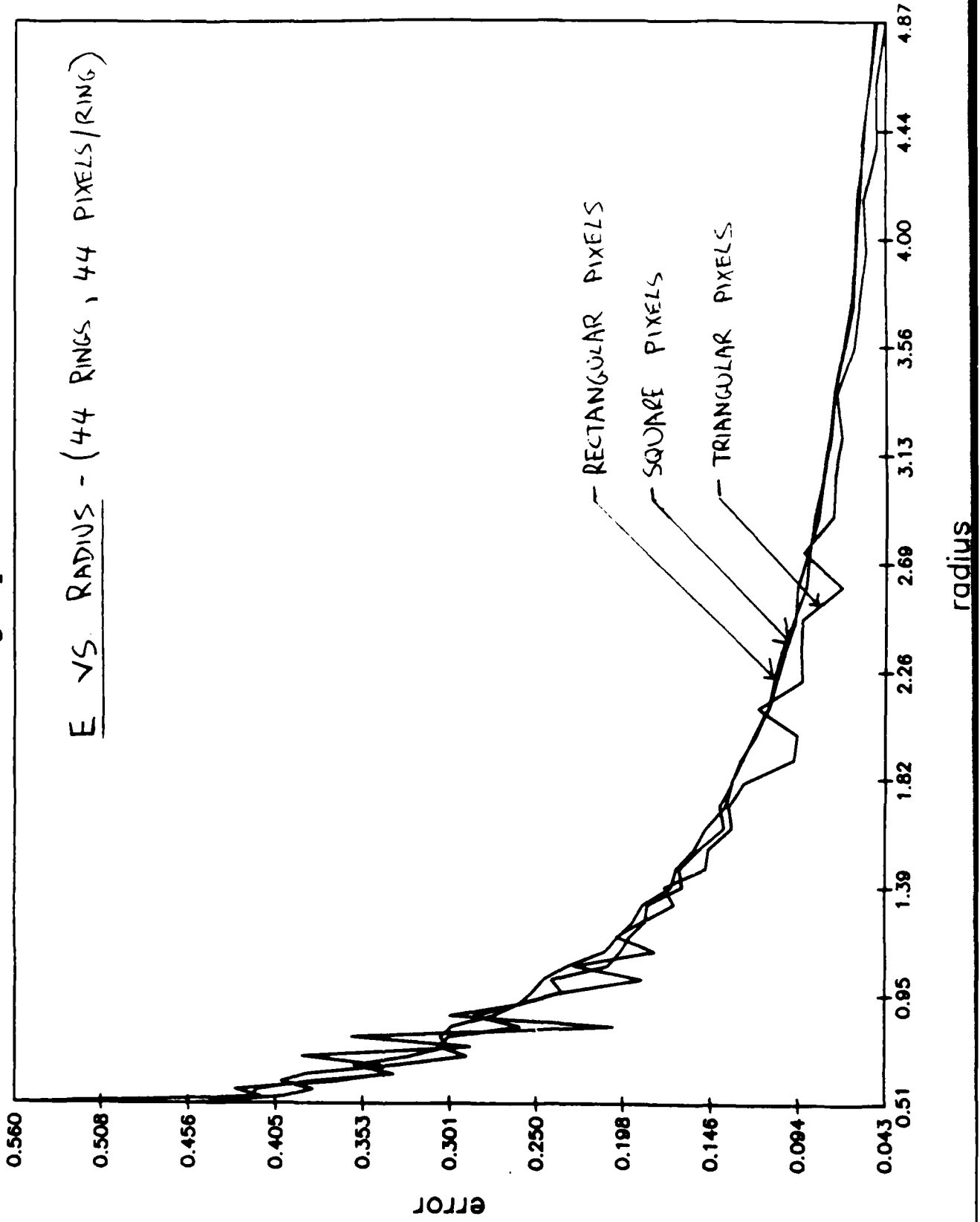
graph seven



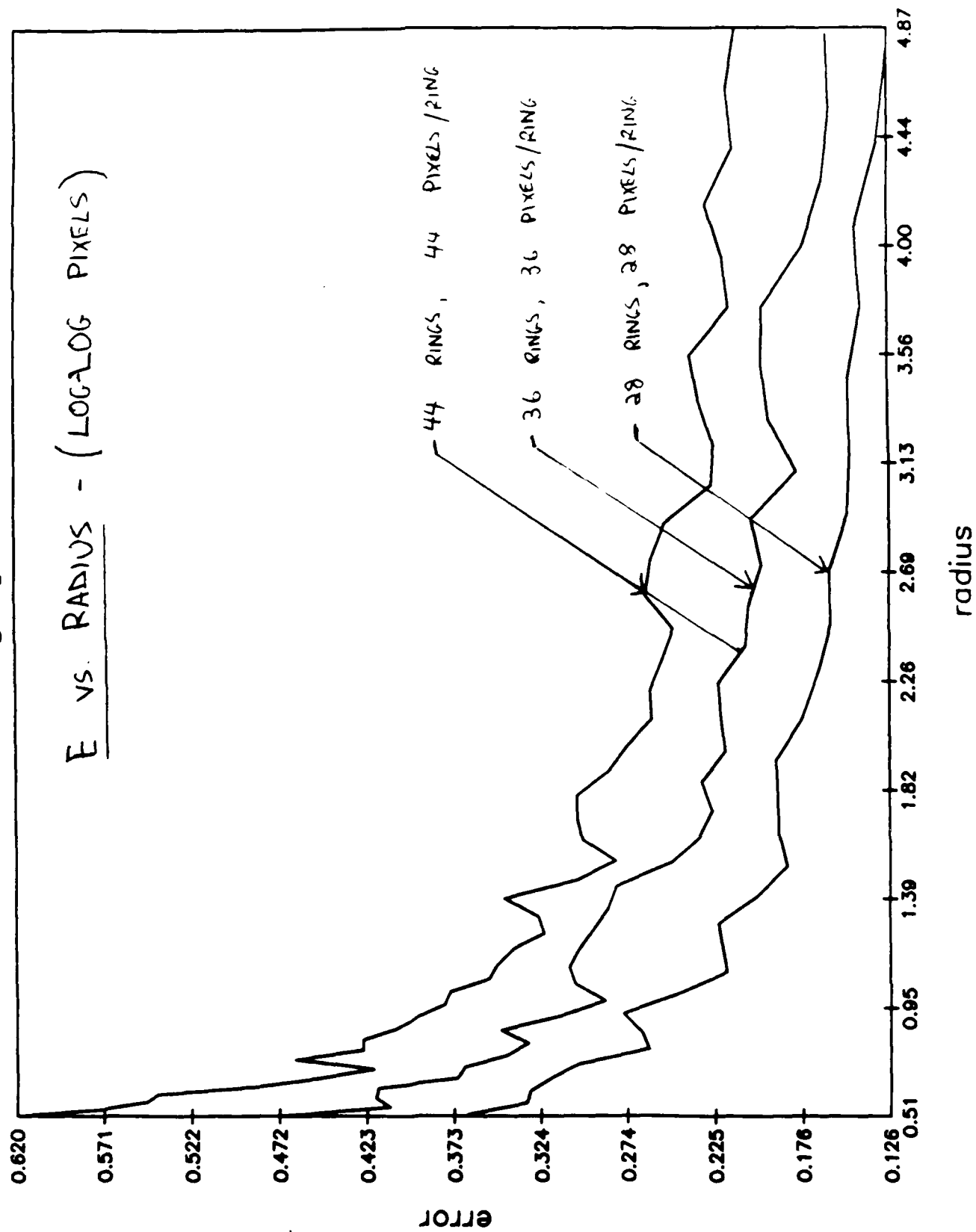
graph eight



graph nine



graph ten



**UNIVERSITY OF VIRGINIA**  
**School of Engineering and Applied Science**

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 560. There are 150 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 16,400), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Physics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

100

100-100000-100

1990